SÓLO

PROGRAMADORES

Revista especializada para usuarios de PC

AÑO 1 Nº 3
1250 PTAS.



Curso de programación de C++

Rutina para la reducción de colores

DE REGALO
La 1ª entrega
del Compilador
SYMANTEC
C++ 6.0

Programación de macros para Excel

Y además:

- DLLs para OS/2
- Métodos de ordenación de datos
- Reconocimiento de objetos en 2D
- Monitor de memoria



PROGRAMACIÓN BAJO MS-DOS EN 32 BITS







CARTA DEL DIRECTOR

El ABC DEL PROGRAMADOR

Parece ser que el número anterior de SÓLO PROGRAMADORES sorprendió agradablemente a más de uno, y digo esto por los comentarios que empiezan a circular en las BBS más conocidas. Las cartas que han llegado a la redacción resaltan sobre todo el tema del Linux, que hubiera sido prácticamente imposible de conseguir por otros medios si no se hubiera incluido en la revista, principalmente por su tamaño. El hecho es que el UNIX tiene en nuestro país más *adeptos* de los que la gente piensa y por ello comenzaremos un curso del mismo (basado en el Linux) a partir del próximo mes.

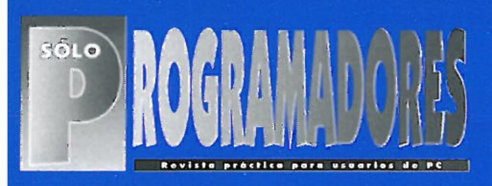
Este mes, merced a un acuerdo editorial con SYMANTEC, SÓLO PROGRAMADORES ha conseguido los derechos de publicación de la versión 6.0 del compilador completo de C++, sin duda uno de los mejores del mercado. El objetivo es proporcionar a nuestros lectores, sin costo alguno, un compilador de C++ para poder seguir los ejemplos y programas publicados en C y C++ en la revista. Debido al extraordinario tamaño del programa (18 discos de alta densidad), lo dividiremos en varias entregas para ofrecer mes a mes todas las partes y librerías del programa, incluyendo el mes próximo un artículo sobre el paquete completo SYMANTEC C++ como primera toma de contacto.

En noviembre seguimos con la línea de artículos y cursos comenzada en números anteriores, teniendo como objetivo *frenar momentaneamente* para repasar conceptos y *sentar ideas.* Esa ha sido la consigna bajo la cual se ha realizado la actual entrega de los cursos de SÓLO PROGRAMADORES. Llegó el momento de ponerse *manos a la obra.* Por supuesto, no faltan los artículos *duros* de rigor como la Programación en Modo Protegido del 386 o el Monitor de Memoria para los más audaces. El artículo sobre las Macros en Excel pretende ilustrar el uso de estas potentes herramientas a la hora de trabajar con este programa, una de las hojas de cálculo más utilizadas en la actualidad.

Como cada mes a estas alturas, repasamos las cosas que habíamos pensado incluir y, finalmente, se quedaron en el tintero. Y es que el tiempo, lejos de ser nuestro aliado, se convierte en un peligroso competidor que corre cada vez más rápido.

En el próximo número abordaremos nuevas ideas y servicios a nuestros lectores (la BBS propia de SÓLO PROGRAMADORES, paciencia por el momento), así como los resultados de la encuesta publicada en el número uno. Les dejamos con los primeros disquetes del paquete de SYMANTEC. Hasta la próxima.

MARIO DE LUIS



NOVIEMBRE 1994. Número 3
Es una publicación de



COMMUNICATIONS, S.R.L.

Editor
Antonio M. Ferrer Abelló
Directora Comercial
Carmina Ferrer
Publicidad
Magdalena Pedreño Llorente

••••••

Director
Mario de Luis García
Redactor Jefe
Carlos Doral Pérez
Coordinador técnico
Miguel Angel Alcalde

Colaboradores
María Gil, Luis F. Fernández, Ricardo Lazo, David Rubio, Bernardo García, J. Ramón Lehmann, Oscar García, Agustín Guillén, Rafael Alcalde, Francisco Martín, Ignacio Cea, César Astudillo.

Edición técnica
David Rubio
Maquetación
Alejandro Guindulaín
Asesor Técnico de Autoedición
Antonio López Areosa
Servicios Informáticos
Digital Dreams Multimedia, S.L.
Ilustraciones
Miguel Alcón, Nacho Morales
Secretaría de Redacción
Consuelo Jiménez
Suscripciones
Erika de la Riva
Redacción, Publicidad y Administración
C/ Marqués de Portugalete, 10
28027 MADRID
Telf.: 741 26 62 / Fax: 320 60 72
Filmación
Grafoprint S.L.
Impresión
Gráficas Reunidas, S.A.
Distribución
MIDESA

Depósito legal: M-17806-1994
ISSN: 1134-4792

SUMARIO

SYMANTEC C++ 6.0

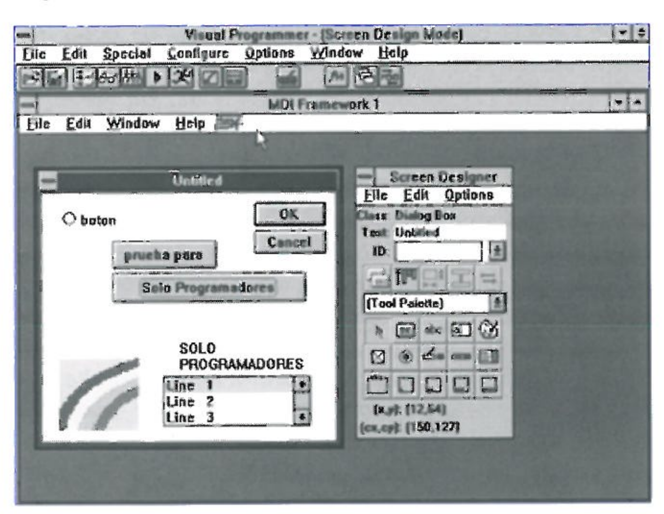
a partir de este mes.

SOLO PROGRAMADORES regalará el compilador Symantec C++ 6.0, por gentileza de SYMANTEC Corporation. Debido a la gran cantidad de discos que componen el paquete, la entrega se efectuará durante varios números de la revista.

SYMANTEC C++ 6.0

Es un sistema de desarrollo para los programadores que trabajan con la familia de procesadores 80X86 de Intel y que utilizan los sistemas operativos DOS o Ms-Windows.

Este compilador es el resultado de una profunda revisión del Zortech C++. Incluye un entorno integrado en Ms-Windows, soporta trabajo en grupo, posee nuevas librerías de objetos y permite la programación en 32 bits bajo DOS o Windows 3.1 con Win32.



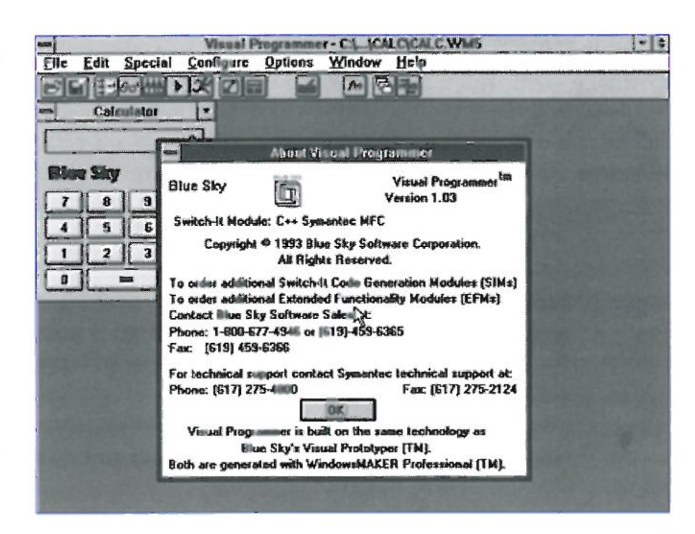
CONFIGURACION MINIMA.

Los requisitos para poder utilizar Symantec son los siguientes:

- Procesador 80386 o superior.
- Microsoft Windows 3.1 o posterior.
- 4 MB disponibles de RAM.
- 10 MB libres en el disco duro.

CONTENIDO DE LOS DISQUETES.

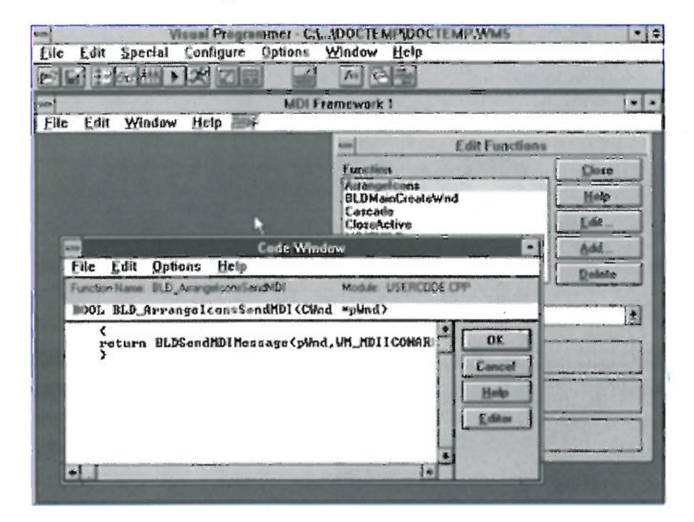
Este mes se incluye en los disquetes de SOLO PROGRAMADORES Visual Programmer, un generador de aplicaciones que se puede emplear para crear prototipos, chequear y generar aplicaciones en C++ que aprovechan los recursos de Microsoft Foundation Class Library 2.0.



Visual Programmer posee los elementos familiares del interfaz de usuario de Ms-Windows, tales como menús desplegables, barras de herramientas, cajas de diálogo, y otros elementos, como objetos en 3D que se pueden incluir instantáneamente en la aplicación que se esté desarrollando.

Para instalarlo se debe ejecutar desde Ms-Windows, el fichero INSTALL.EXE que se encuentra en el primer disquete de la revista. Para completar con éxito la operación se debe disponer al menos de 6 MB libres en el disco duro, necesarios para la descompresión, ocupando finalmente sólo 2 MB.

Tras finalizar este proceso, se puede ejecutar el programa VPROG.EXE que se encuentra en el directorio del disco duro ...\sc\bin. Así mismo se pueden cargar dentro de esta aplicación los ejemplos localizados en el directorio ...\sc\mfc\samples.



NOTICIAS



VERSIONES EN VARIOS IDIOMAS DE PRODUCTOS TCP/IP DESKTOP DE NOVELL

Novell ha presentado en varios idiomas los productos TCP/IP desktop LAN WorkPlace 4.2 y LAN WorkGroup 4.2. A partir de ahora los usuarios podrán disponer de estos paquetes en francés, español, portugués y alemán. También habrá una versión en japonés, distribuida a través de la compañía Novell K.K. en Japón.

TCP/IP es un protocolo que permite la interoperabilidad entre sistemas diferentes. LAN WorkPlace conecta a los usuarios de MS-Windows y DOS a miniordenadores y mainframes UNIX. Por su parte, LAN WorkGroup, se encarga de proporcionar a las redes NetWare acceso a sistemas UNIX desde su ordenador.

Se calcula que en 1.994 habrá un total de ocho millones de nodos de TCP/IP en todo el mundo.

RLN APPLICATION SERVER DE DCA

El RLN Application Server ha sido presentado por DCA. Se trata de un software para redes Novell, que se integra con Remote Lan Node (RLN) para permitir una mejora de la ejecución en conectividades remotas para aplicaciones intensivas de ficheros. El centro tecnológico de este producto es Win View for Networks, licenciado por Citrix Systems Inc. Se persigue un acceso remoto más rápido para aplicaciones que requieran alta velocidad, así como para grandes sistemas de bases de datos. La combinación de DCA RLN y de RLN Application Server hace posible que los usuarios remotos accedan a aplicaciones cliente/servidor e intensivas basadas en red.

RLN es un software remoto cliente/servidor que maneja simultáneamente hasta 16 usuarios a distancia. Procesa aplicaciones remotas emulando una tarjeta de red en el PC. De este modo, los usuarios pueden operar como si estuvieran físicamente enganchados a la red.

OLICOM RECIBE LA APROBACION DE NOVELL PARA LA CERTIFICACION LAN INTERNA

La compañía Olicom ha recibido por parte de Novell la aprobación a su laboratorio de tests para la certificación interna de compatibilidad con Netware. Esta autorización significa que la propia Olicom podrá dilucidar si su producción es o no compatible con Netware. De este modo será posible que se comercialicen con mayor celeridad productos con la certificación Novell.

La Novell Certification Alliance (NCA) es la encargada de autorizar a determinados fabricantes para que realicen sus propios tests de compatibilidad con NetWare. La NCA trabaja en estrecha colaboración con varios fabricantes, y establece los procesos y métodos detallados utilizados en dichos tests.

NUEVOS MODELOS DE SERVIDOR PARALELO CORPORATIVO S/390 9672 DE IBM

IBM ha anunciado la aparición de seis nuevos modelos de Servidores Paralelos Corporativos S/390 9672. Dichos servidores han sido diseñados para cualquier tipo de aplicación informática de empresa, desde la gestión contable hasta los servidores de vídeo. Estos nuevos modelos amplían la escalabilidad de los procesadores paralelos S/390, para atender a aquellos usuarios que trabajen con mainframes de gama media. Se pretende con ello que se produzca una evolución hacia los mainframes basados en microprocesadores.

Para convertir los nuevos sistemas de bajo coste en

NOVELL

plataformas que puedan emplearse como servidores en entornos cliente/servidor o como procesadores autónomos, se han lanzado nuevas versiones (o releases) de los sistemas operativos para grandes sistemas de IBM (VSE, VM, MVS), y se ha reducido el precio del software.

ANIXTER DISTRIBUYE LA NUEVA FAMILIA DE HUBS PARA CONMUTACION ETHERNET

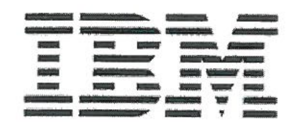
Anixter internacional distribuirá desde ahora la familia de hubs para conmutación Ethernet SynOptics LattisSwicht System 28000. Esta serie ofrece a los administradores de redes flexibilidad a la hora de mejorar su rendimiento, ya que aporta un ancho de banda escalable.

Estos distribuidores de conexionado permiten aumentar la velocidad de casi cualquier diseño de red. También se mejora el control administrativo. La familia de hubs LattisSwitch admite Ethernet a 10 y 100 Mb. por segundo a través de fibra óptica y cableado de par trenzado no blindado (UTP). Se pueden configurar en cascada para permitir la creación de estructuras de red en configuraciones de mayor tamaño.

WINDOWS NT Y UNIX TRABAJARAN JUNTOS DE MANERA TRANSPARENTE CON LA SOLUCION NETWORK FILE SYSTEM DE INTERGRAPH

La solución PC-Network File System (NFS), así como el DiskShare, proporciona herramientas estándares de interoperatividad para un entorno mixto de sistemas UNIX y Windows NT.

La solución NFS incluye dos productos: PC-NFS y DiskShare, ambos para Windows NT. El servidor X Windows Systems se denomina eXalt. Estos productos pretenden fomentar la interoperatividad y la comunicación abierta entre Windows NT Advanced Server de Microsoft y sistemas operativos UNIX.



Los productos cliente/servidor NFS están basados en los estándares Open Network Computing/NFS (ONC/NFS) e integran sistemas Windows NT y Windows NT Advanced Server en redes multifabricante cliente/servidor.

IRMA WORKSTATION PARA OPEN SYSTEMS Y PARA AS/400 DE DCA

DCA ha presentado recientemente las versiones de su gama Irma Workstation para Open Systems y para AS/400. La primera ha sido diseñada para que los usuarios puedan acceder a una serie de hosts tales como UNIX, DEC y Hewlett-Packard. Este acceso puede ser también simultáneo. Irma Workstation para Open Systems soporta DCA QuickApp para Windows 2.1, así como APIs, HLLAPI, DDE y CASL.

También está disponible Irma Worstation para AS/400, versión 2.0. Gracias a este software los PCs pueden acceder a hosts IBM AS/400. IWAS/400 aporta un soporte TN 5250 para redes TCP/IP, además del SNA. De este modo se puede escoger cualquier conexión e intercomunicar diferentes tipos de red. Incorpora además un transferencia de ficheros y soporte para SNA Server de Microsoft Windows NT y Text Assist.

INTEL REDUCE EL PRECIO DE SU PROCESADOR DEDICADO INTEL386 EX

Intel Corporation reducirá desde ahora los precios de su familia de procesadores dedicados Intel386 EX. Este procesador, del que se empezaron a producir muestras en abril, forma parte de una gama de aplicaciones de esta compañía, algunas de las cuales ya están en producción.

Dicha reducción, que será efectiva desde el 1 de octubre, disminuirá el precio de estos procesadores de $52 a $29 por unidad.

El Intel386 EX se basa en el núcleo del 386SX y proporciona un motor compatible con DOS con características adicionales para los sistemas dedicados. Dichas características son las siguientes: ocho selecciones de chips programables, dos canales DMA, tres canales timer/counter, dos controladores de interrupciones, dos canales SIO asíncronos, un canal dúplex SIO síncrono, refresco de DRAM y pseudo SRAM, contador Watchdog, y escaneo de carpeta JTAG entre otras. ■





BREVES

NOVELL ANUNCIA TUXEDO/DCE

Novell Inc. ha anunciado la disponibilidad de la extensión TUXEDO/DCE a su monitor de proceso transaccional. Se trata de un entorno de desarrollo que crea aplicaciones distribuidas para empresas.

NOVELL COMERCIALIZA NETWARE 4.02

Novell está distribuyendo la versión más moderna del sistema operativo Netware 4. Netware 4.02 ha simplificado su utilización y reducido el coste de implentación, funcionamiento y mantenimiento.

DCA PROVEE ACCESO A INTERNET CON LA ÚLTIMA VERSIÓN DE CROSSTALK

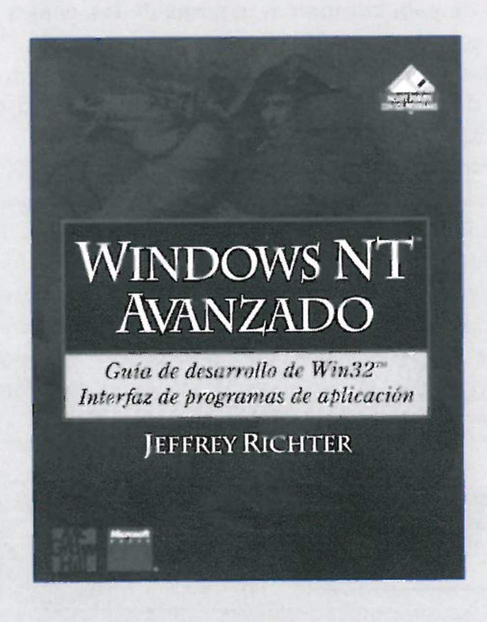
DCA ha anunciado la versión 2.2 de Crosstalk para Windows. Este paquete de comunicaciones ha sido completamente modernizado, con lo que se pretende que el acceso y la navegación por Internet se haga más sencilla. Este programa posee un Internet Helper que configura Crosstalk para conectar al Servicio Internet y acceder a aplicaciones como el correo electrónico, la transferencia de ficheros y la búsqueda de información.

LIBROS

WINDOWS NT AVANZADO

Este libro, destinado a los diseñadores corporativos y comerciales, pretende convertirse en una fuente de información fiable sobre las áreas del núcleo del sistema operativo Windows. El autor se ha centrado en las novedades y principales puntos de interés que presenta este paquete de desarrollo de software. Se suministran además 25 programas-ejemplo para hacer un uso práctico de dichas características.

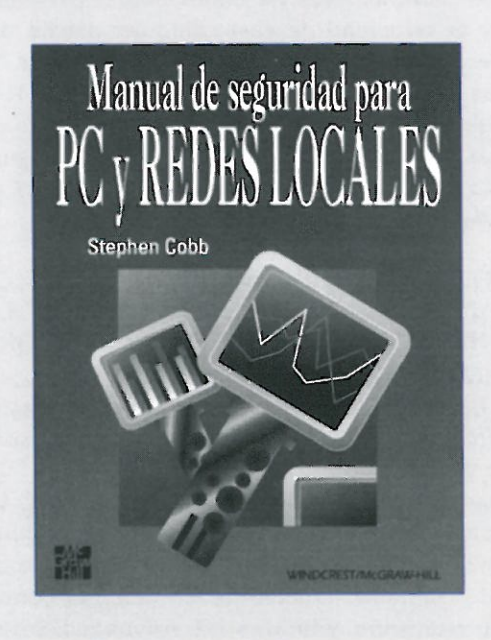
Editorial: McGraw-Hill (91) 372 84 11
Autor: Jeffrey Richter
645 Páginas
Precio: 5.000 pts.



MANUAL DE SEGURIDAD PARA PC Y REDES LOCALES

Para todas aquellas personas para las que la seguridad de sus datos es una preocupación, este manual puede resultar de gran ayuda, ya que ofrece una serie de métodos efectivos para proteger el ordenador tanto de curiosos intrusos como de fallos del equipo. Se le ha dado a esta guía un enfoque práctico y accesible, para que el salvaguardar su ordenador o red de piratas, virus, competidores, incendios, despistes, etc. no suponga un problema.

Editorial: McGraw-Hill (91) 372 84 09
Autor: Stephen Cobb
626 Páginas
Precio: 4.800 pts.





LA AMENAZA DE GOLEM

En varias leyendas hebreas de la Edad Media, el golem aparece como un hombre de barro al que un sabio judío insuflaba vida sin ingeniería genética ni nada. Instrucciones de instalación: ir introduciendo en la boca de la efigie pequeños trozos de papel con las letras que forman uno de los nombres de Dios (yo no intentaría el experimento si no es en compañía de un adulto responsable). El golem solía ser un perfecto sirviente. Su único defecto: ejecutar los comandos de su amo de una forma mecánica y tercamente literal (¿le suena?). Las consecuencias de este comportamiento podían ser cómicas o llevar a un desenlace lleno de hemoglobina, como en el cuento del rabino Ben Bezulel de Praga, relato apropiado para mantener a los peques bien arropados y calladitos en tiempos en que aún no había cine gore. El monstruo de Frankenstein viene a ser una encarnación posterior de este mito medieval.

Seguro que quien haya escrito más de cinco líneas de código habrá compartido en algún momento la frustración del rabino de Praga. No basta con dominar el lenguaje que entienden los oídos de barro del golem: hay que saber pensar como él, elaborar modelos para la realidad en los términos de su obtuso y literal entendimiento. Algo más fundamental y de mayor calado que la simple codificación en un lenguaje.

Hasta los usuarios admiten tarde o temprano el hecho de que el ordenador no es más que un golem. No es que el estúpido ordenador haya borrado tu trabajo, guapo, el pobre sólo sigue órdenes. Pero nuestra misión como desarrolladores no es otra que ésta: aislar al usuario de la obstinada inhumanidad de su máquina, hacer de embajadores entre el mundo de la computación y el de los problemas humanos. El usuario está demasiado ocupado para molestarse en pensar como el golem. Nos paga a nosotros para eso. Hasta que nos dimos cuenta, el uso del ordenador para resolver los problemas de la gente corriente estuvo contenido por una barrera de pánico, y con razón: los interfaces de usuario no estaban concebidos a escala humana. Y no se trata de "hacer programas para tontos", expresión utilizada a menudo como única guía de usabilidad. Aunque esta consigna puede ser de ayuda, apesta a displicencia y a torre de marfil. Los que queremos vivir de esto no nos podemos permitir ya esta actitud. De lo que se trata es de darle al programa una metáfora coherente, algo que le presente ante el usuario como una herramienta de este mundo, plegada a las leyes del sentido común. El procesador de textos ofrece una metáfora de máquina de escribir. Esto no ha sido así siempre: ¿Recuerda EDLIN? La hoja de cálculo y sus celdillas, la base de datos y sus fichas, el programa de dibujo y su botecito de pintura, ofrecen otras metáforas olvidadas por lo familiares. Ahora, ordenadores y usuarios esperan que inventemos nuevas metáforas que les ayuden a acercarse. El usuario merece que su programa sea tan sencillo, intuitivo y previsible en su comportamiento como un martillo lo es para el carpintero. Las nuevas e impresionantes prestaciones no añadirán calidad a nuestros programas si no las integramos en la metáfora con accesibilidad y armonía.

Esta tarea exige algo más que simple aptitud para programar: también precisa creatividad. De la misma naturaleza que la que necesita el director de cine para llegar a su público o el publicitario para concebir un mensaje para su campaña. Todo programa se puede mejorar poniendo en juego habilidades creativas. Y no hablo de iconitos horteras y colorines que sólo sirven para despistar. Hablo de aumentar el ancho de banda de la comunicación entre el usuario y su programa, con funcionalidad, con inmediatez, sin estridencias. Para eso sí que hace falta pensamiento creativo y buen gusto. El desarrollador que renuncia a usar la imaginación da validez al tópico que tanto nos perjudica: el de que las máquinas deshumanizan. Para caja tonta, el ordenador. Nuestro oficio consiste en ponerle el duende que le falta, no en contagiarnos de su estupidez.

Por mucho que algún día las máquinas se programen solas, nosotros seguiremos teniendo trabajo. Los únicos desarrolladores condenados a la extinción serán aquéllos que ignoren el antropocentrismo del nuevo software: quienes no sean capaces de desarrollar su cerebro izquierdo para pensar como el golem, y al mismo tiempo potenciar su cerebro derecho para pensar como los usuarios. Este pensamiento doble es el único que puede dar continuidad a nuestro papel de médiums entre el hombre y la máquina.



ENTREVISTA

SHAREWARE

SHAREWARE
PRUEBA ANTES DE COMPRAR

En vista de la difusión y el éxito que el shareware tiene en Estados Unidos, la empresa española Friendware se ha decidido a traerlo a España, para que los usuarios puedan probar los programas a precios muy reducidos, antes de realizar un desembolso importante en los mismos. Juan Díaz-Bustamante y Luis Alarcón, responsables de esta empresa, explican en qué consiste el negocio del shareware.

Aunque la mayoría de los lectores estarán al corriente no se ha querido obviar esta cuestión tan básica.

¿Qué es el shareware?

El shareware es una técnica de márketing. Las empresas de shareware tienen un sistema de distribución diferente al de las casas comerciales. Lo que hacen es promocionar sus productos mediante shareware. Lo que se distribuye, lo que se vende, no es el programa, sino un disquete, que incluye los costes de duplicación, de distribución, de embalaje y de envío. En el caso de un juego, por ejemplo, si tiene varios escenarios, el disco ofrece uno. Si ese programa interesa, se han pagado 800, 700, 600 pts., y se puede acudir a la versión registrada, que ya cuesta alrededor de 6.000. El concepto de shareware tiene un lema en inglés, que es *try before you buy*, prueba antes de comprar, y ése es el secreto del shareware. En teoría, todos los programas deberían utilizar este sistema, y de este modo se evitarían muchas desilusiones por parte de la gente, que a veces se compra un juego que cuesta 10.000 pts. y cuando lo carga descubre que no le gusta.

¿Como surgió el shareware?

A través de estudiantes americanos que comenzaron a sacar juegos, a distribuirlos regalados y demás. Se empezaron a solicitar versiones más



Luis Alarcón y Juan Díaz-Bustamante, los introductores del shareware en España.

avanzadas, más completas, etc. Surgió como un juego y se ha convertido en un negocio multimillonario.

Algo muy importante es que en el shareware se da una relación directa entre el programador y el comprador del programa, que no se da con el resto de los programas comerciales. Los programadores se sienten muy cómodos con el tema del shareware. Normalmente las empresas fuertes están formadas por programadores, no por hombres de negocios que forman una empresa de shareware. Todos han empezado con un programa que funcionó hace tiempo, al que siguieron otros. De momento, todo lo que traemos es de Estados Unidos, pero tenemos pensado, a corto-medio plazo, desarrollar shareware español y ya estamos hablando con programadores españoles. Hay programadores de mucha calidad en España, que hasta ahora no habían conseguido introducir sus programas en un sistema de distribución que pudiera hacer que esos programas se conociesen fuera. De este modo, los programadores no obtenían ningún reconocimiento, nadie los conocía. Con el shareware, ellos siguen manteniendo el copyright, se dan a conocer. En estas versiones, además del logotipo de la empresa aparece el nombre de la casa programadora, de manera que ésta conserva su autoría.

¿Cómo está el mercado español en lo que a shareware se refiere?

En pañales. Notamos un desconocimiento muy grande por parte de la gente. Además de vender tenemos que realizar una labor casi didáctica, y explicar lo que es el shareware. En Estados Unidos todo el mundo sabe que con 800 pts. sólo se están pagando los costes de distribución, de duplicación, etc.

¿El precio de los disquetes cubre exclusivamente los gastos?

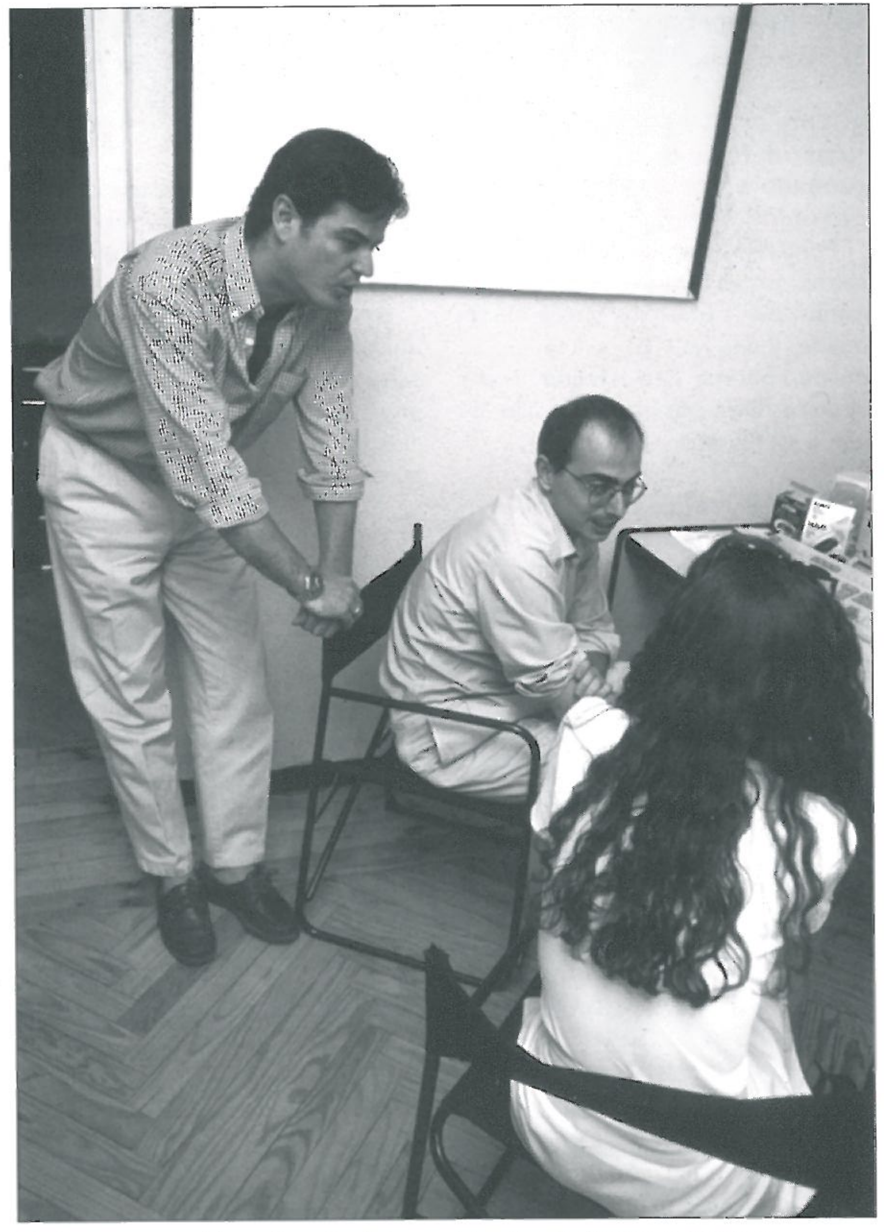
Básicamente sí, aunque hay un pequeño margen de beneficio, como es natural.

¿Cómo se controla el período de tiempo del que dispone una persona para emplear el shareware, antes de registrarse?

De ninguna manera. Hay que distinguir entre lo que es el shareware de juegos y el de utilidades. En el de juegos se trata de distintos niveles o escenarios a los que se puede acceder. En el caso de las utilidades siempre va a aparecer un monigote en el cual se advierte que es una versión de prueba, y que si se considera útil, hay que registrarse. Se tiene la obligación moral de hacerlo. La versión registrada siempre da más, tanto en utilidades como en juegos. Las versiones shareware de juegos, al menos en nuestro caso, son utilizables en un 95%. En Estados Unidos este tema está legislado, y el usuario tiene un periodo que normalmente es de 30 días. Al cabo de ese tiempo hay que registrarse, y si no gusta la versión shareware, sacarla.

¿Hay algún tipo de legislación en nuestro país que regule el mercado del shareware?

Nada. Bueno, los programas poseen copyright, y la versión shareware también. Quizá haya un poco



El shareware surgió gracias a los intercambios de estudiantes americanos.

de peligro al trasladar este sistema al mercado español.

¿Cómo están tan seguros de que esa "obligación moral" que tiene la persona de registrarse transcurrido el período de prueba será cumplida?

No estamos nada preocupados por eso, por lo siguiente: por el propio programa de shareware, que si es lo suficientemente bueno, va a conseguir que la gente se registre. Si alguien juega un nivel de un juego y sabe que registrándose va a tener seis niveles más, y el juego es buenísimo, no se va a quedar siempre en el mismo nivel. Si los juegos no fuesen buenos estaríamos preocupadísimos.

¿Cómo se puede registrar un usuario después de que ha probado un programa y le ha gustado?

Poniéndose en contacto con nosotros. En el disquete shareware figuran todos nuestros datos claramente, y una hoja de pedido que se puede imprimir directamente desde el disco. Otra posibilidad es llamarnos o escribirnos.

¿Cómo se realiza la labor de recopilación y distribución de estos programas shareware?

Nosotros tenemos firmada una licencia de distribución exclusiva en España con las casas americanas, y ellos nos envían los masters de las versiones. Cuando sale una novedad, ellos nos la mandan rápidamente, a través de una red. Nosotros nos encargamos de la duplicación, de la elaboración y traducción de los manuales, y de las cajas. Por lo que se refiere a los propios programas, estamos traduciendo a unas casas de utilidades, para salir dentro de unos meses con utilidades en español. El resto de los programas está en inglés, pero se trata de juegos sencillos, muy adictivos y fáciles de usar, por lo que para su desarrollo no es necesario ningún conocimiento de inglés.

Como distribuidor de shareware estás obligado a significar en tu publicidad que este sistema permite probar los programas por una pequeña cantidad de dinero, antes de pagar al autor.

La experiencia del shareware, que tan bien funciona en EE UU, ¿Se ha intentado en algún país europeo?

Sí, funciona en Inglaterra, pero quizá el caso más afín al español sería el de Italia. En Italia, una empresa consiguió la distribución exclusiva de las versiones. Lleva dos años funcionando y le va muy bien.

¿Cómo se hacen llegar las versiones shareware a los potenciales usuarios?

Bien a través de revistas, publicidad, entrega de disquetes a publicaciones para que los inserten en sus números, etc., o bien a través de catálogos y mailing. Ahora vamos a empezar a introducir algo que se denomina rack vending. El rack es el expositor, en el que se contienen 10 o 76 disquetes shareware en cajas de plástico o cartón. Estos expositores se colocan en cadenas de tiendas, grandes superficies, etc. Las tiendas reciben un porcentaje sobre la venta, y no tienen que hacer nada. Este es un sistema que funciona extraordinariamente bien fuera, y que puede funcionar bien aquí porque los precios son fenomenales. En el shareware que va dentro figura la publicidad de todos los demás productos de las casas americanas, y dónde pueden adquirirse

¿Es siempre un dsiquete el soporte del shareware?

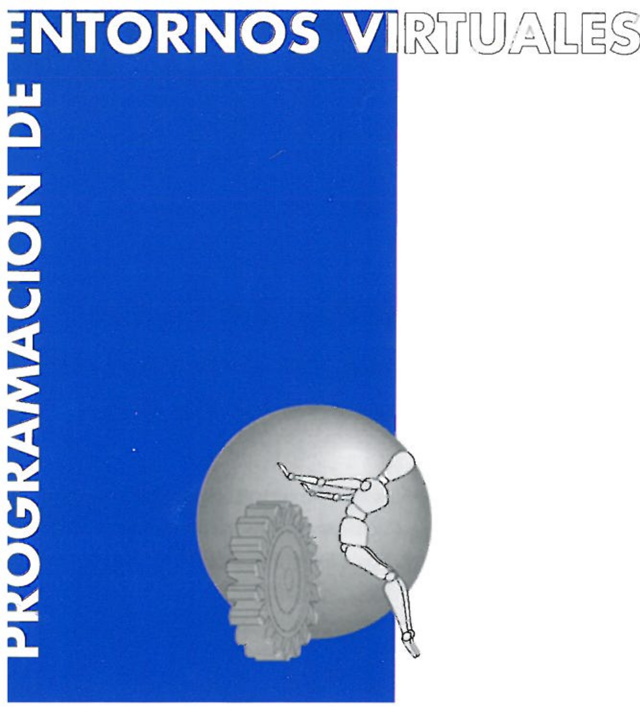
Estamos pensando en algún tema en CD-ROM, pero a medio plazo. Sería un CD-ROM promocional, con 12 juegos buenos, pero no queremos despersonalizar los juegos introduciéndolos en un CD, a no ser que uno de ellos se destaque como estrella dentro del mismo. Una posibilidad sería un CD-ROM con todos los programas de Apogee. ■

Esta joven empresa se plantea como objetivos principales dar a conocer y extender este sistema en España, y potenciar el shareware realizado por programadores nacionales, para alcanzar la calidad y difusión que esta técnica de márketing tiene en otros países.

PROGRAMADORES

A partir del próximo mes, publicaremos en SÓLO PROGRAMADORES nuestra propia bolsa de trabajo. Todas aquellas personas que estén interesadas, deberán enviar su *curriculum vitae* a la dirección:

Sólo Programadores
Referencia: *PROGRAMADORES*
TOWER COMMUNICATIONS
C/ Marqués de Portugalete, 10 Bajo
Madrid 28027



MOVIMIENTO FLUIDO EN 3D

Luis Fernando Fernández

En un principio, este artículo debería estar dedicado a la realización de una demostración de las posibilidades que nos ofrecen los conocimientos adquiridos en anteriores artículos, pero aún faltaba por atar un cabo fundamental, el de conseguir un movimiento suave.

EL ACCESO A VIDEO

El momento de la verdad ha llegado, ya se pueden definir objetos, situarlos en el espacio, hacer que giren y el observador es capaz de moverse en ese mundo recién creado pero...

La pantalla parpadea y el movimiento es lento, la desesperación se apodera del programador que ve como la culpa no puede ser suya, él ha seguido paso a paso lo que le indicaron, y por fin surge la pregunta ¿de quién es la culpa? La respuesta es simple, el modo en que se está accediendo a video desde C.

Hasta ahora, para dibujar cada imagen, se borraba en primer lugar la pantalla. Al intercalar una pantalla negra entre cada imagen se produce un molesto parpadeo. Este problema tiene una solución muy simple; consiste en crear las imágenes en RAM en lugar de en memoria de video, una vez compuesta la imagen, ésta se vuelca a video, a continuación se borra la pantalla de RAM, con lo cual ese proceso se convierte en algo transparente al usuario y se vuelve a repetir el proceso de creación y volcado a video (ver figura 1).

Además la resolución que se ha elegido hasta ahora (640x480) es algo grande para lograr un movimiento rápido. Se va a reducir su tamaño considerablemente para acelerar los procesos de volcado. De nuevo aparece un problema, si se reduce el ancho horizontal a la mitad, esto es, 320 pixels de ancho, el modo de video por defecto de la VGA que soporta ese ancho, tiene un alto de pantalla de 200 pixels. El ratio del monitor es de 3/4, esto es, para que los pixels sean cuadrados, por cada 3 pixels de alto debe haber 4 pixels de ancho. En el modo 640x480 esto se cumple, lo podemos ver mediante una simple igualdad:

$$\frac{640}{4} = \frac{480}{3}$$

En cambio en el modo 320x200 no se cumple:

$$\frac{320}{4} <> \frac{200}{3}$$

Los pixels son ligeramente más altos que anchos, por tanto si se dibuja una circunferencia en pantalla, ésta

tendrá aspecto de elipse. Aunque la distorsión es bastante pequeña, es lo suficientemente grande como para que pueda ser apreciable a simple vista. Para solventar dicho problema existen modos no documentados de la VGA que consiguen una variedad de resoluciones bastante amplia, una de ellas es 320x240 que tiene la particularidad de tener los pixels cuadrados. Algunos de los modos no documentados fallan en determinadas tarjetas, no es el caso de este que se va a utilizar, ya que ha sido largamente testeado e incluso se ha incluido en algunos juegos comerciales. A esta serie de modos de video no documentados de la VGA se la conoce con el nombre genérico de modo X, al margen de la resolución.

EL MODO X

Como el objetivo de esta sección son las 3 dimensiones, sólo se va a explicar someramente en qué consiste y las posibles ventajas que se van a obtener con su uso. Para un análisis a fondo, probablemente se dedicará al menos un artículo en próximos números de la revista.

Para activarlo hay que indicar al sistema operativo que se quiere usar el modo de video 320x200, esto se realiza mediante las rutinas gráficas del C o bien mediante la interrupción 10h desde Ensamblador, colocando en el registro AX el valor 13h.

En la tarjeta VGA con el modo gráfico 320x240, los pixels son cuadrados

Una vez en ese modo de video, hay que efectuar unos cambios en el registro de datos del controlador del tubo de rayos catódicos y se accederá al modo no documentado deseado. Pero como eso no es un tema que incumba el programador de 3D, se incluye en el disco que acompaña a la revista, una librería para activar y desactivar el modo de video desde C.

El nombre de dichos procedimientos es:

```
int svmode();     //Inicializa el modo de
                  video (set video mode)
void rvmode();    //Finaliza el modo de
                  video (reset video mode)
```

El primero de ellos devuelve la dirección de memoria donde se debe crear la nueva imagen (zona de memoria no visible en pantalla) para evitar el parpadeo, mientras que el segundo no devuelve nada, simplemente coloca el modo de video en modo texto.

La desventaja de colocar un modo de video no documentado, es que ninguna de las primitivas gráficas del C funcionará, con lo cual habrá que ir programándolas a medida que avance la serie, ampliando la librería original.

La segunda ventaja que se va a encontrar con el modo X, (la primera es, por supuesto, la posibilidad de trabajar con pixels cuadrados), es que la copia de pantalla no va a estar en RAM sino en la propia memoria de video.

La explicación es sencilla. La tarjeta VGA estándar dispone de 256 Kbytes y al seleccionar un modo de VGA, no SVGA, sólo se puede acceder a esa memoria aunque se disponga de una tarjeta de 512 Kbytes o incluso de 1 Mbyte. Además de esas 256 Kbytes tan sólo se pueden direccionar 64 Kbytes, el resto no es accesible, con lo cual, si se tiene en cuenta que una pantalla de 320x200 pixels con 256 colores ocupa 64000 bytes se llega a la conclusión de que en memoria de video sólo se puede tener una pantalla y la copia para evitar el parpadeo se debe tener en RAM con la consiguiente pérdida de tiempo al tener que estar realizando volcados continuamente a memoria de video.

Al usar el modo de 320x240 pixels con 256 colores, cada pantalla ocuparía 76800 bytes y aparecería un problema serio de almacenamiento, la pantalla no entraría en memoria.

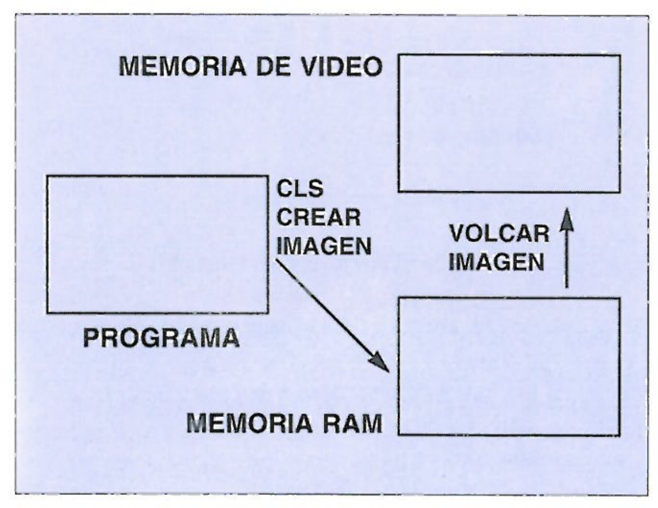
En modo X se resuelve dicho problema gracias a su sistema de almacenamiento, cada pantalla está dividida en cuatro planos, cada uno de los cuales está situado en uno de los cuatro posibles bloques de 64 Kbytes en que se pueden dividir los 256 Kbytes de la memoria de video con lo cual las pantallas ocupan cuatro veces menos.

En la resolución que se va a utilizar aquí, cada pantalla ocupa 19200 bytes y por tanto también se puede tener la copia en memoria de video.

Para visualizar la copia oculta, tan sólo hay que modificar un registro del controlador de rayos catódicos (CRT) indicando la dirección de la VGA que hay que visualizar en pantalla, para ello se incluye un nuevo procedimiento en la librería:

```
int imagen(int);1
```

Este procedimiento recibe como parámetro la dirección de la copia oculta. Dicho valor se obtiene inicialmente del procedimiento *svmode()*, comentado ante-



Figua 1: Creación de una copia intermedia de pantalla para evitar el parpadeo.

riormente, y devuelve la dirección de la nueva página oculta, que es la que inicialmente estaba visible. Este nuevo valor es el que se utiliza en la siguiente llamada al procedimiento cuando se haya creado la nueva imagen. El programa principal se puede ver en el listado 1.

La copia de la pantalla se almacena en la memoria de vídeo de la VGA

El problema de usar nuestro propio modo gráfico radica, como se explicó anteriormente, en que ya no se pueden usar las librerías gráficas del C; por tanto cada vez que se quiera utilizar una primitiva gráfica, como puede ser el dibujo de líneas, hay que implimentársela, bien en el propio C o mejor aún para conseguir una mayor velocidad, en ensamblador.

En cada artículo se explicarán los algoritmos, pero en el disco que acompaña a la revista, se incluirán dichos algoritmos implementados la mayoría de las veces en ensamblador, en forma de librería para que puedan ser usados desde C.

Por comodidad para el usuario, no se explicará el modo de acceder a cada pixel de pantalla en modo X, el proceso será totalmente transparente ya que no viene a cuento en esta serie. Lo que realmente interesa es disponer, como en el caso del presente artículo, de la rutina de dibujar líneas y entender el mecanismo de cálculo de dichas líneas para la posterior implementación de nuevas primitivas gráficas como puede ser el dibujo de polígonos; el resto son ganas de complicarse la vida. De todos modos, para los que puedan estar interesados, esto se detallará en los futuros artículos dedicados al modo X.

LISTADO 1

```
void main()
{
        //Definición de variables
                ...
        página_oculta = svmode();
                ...
        {               //Bucle principal
                        ...
                Creación y dibujo de la imagen
                en página_oculta
                página_oculta = imagen(página_oculta);
                        ...
        }               //Fin del bucle principal
                ...
        rvmode();
                ...
}
```

ALGORITMO PARA DIBUJAR RECTAS

A partir de este momento se considerará que la coordenada (0,0) de pantalla está situada en la esquina superior izquierda del monitor. La coordenada X crece de izquierda a derecha y la coordenada Y lo hace de arriba a abajo. Esto es sólo de cara a la explicación del algoritmo de pintar líneas, ya que de cara a proyectar los puntos 3D en la pantalla de visión se seguirá considerando a la coordenada (0,0) como el centro de la pantalla.

El problema de representar líneas en el ordenador es debido a que el tamaño de los pixels hace que no sea posible dibujar una línea perfecta, siendo necesaria una aproximación pintando en pantalla los pixels que más se aproximen a la recta real. Por supuesto, si se aumenta la resolución la aproximación será mejor (ver figura 2) al aumentar el número de pixels sin aumentar el tamaño de la pantalla (el monitor no crece), sin embargo, los cálculos serían mayores y la velocidad disminuiría considerablemente, y el objetivo que se ha marcado esta serie es el movimiento en tiempo real.

En un primer acercamiento al cálculo de los puntos de la línea, lo lógico sería usar la ecuación de la recta:

$$Y = Pendiente * X + A \qquad (1)$$

donde (X,Y) son los pares de puntos que forman la recta, *A* es el punto de corte de la recta con el eje Y, y la *Pendiente* indica la inclinación de la recta.

Pero como lo que se tiene no es ni la *Pendiente* ni *A*, sino las coordenadas de los extremos de la recta (ver figura 3), hay que calcular previamente dichos valores. Para ello se va a partir de otra expresión de la ecuación de la recta:

$$\frac{Y - Y0}{Y1 - Y0} = \frac{X - X0}{X1 - X0}$$

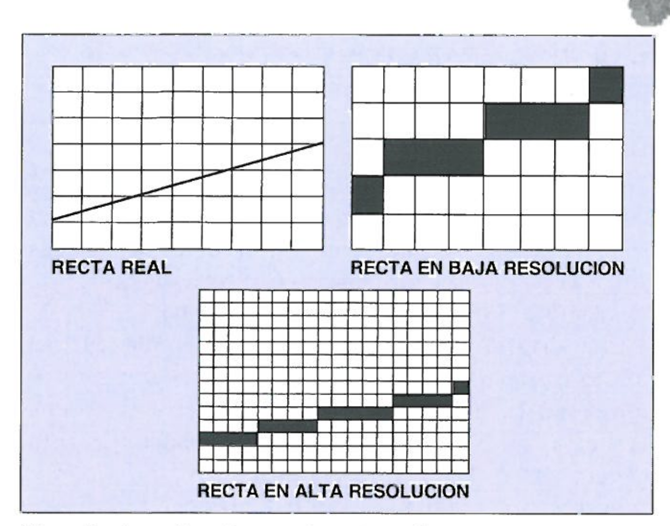
Donde los pares (X0,Y0), (X1,Y1) son los extremos de la recta. Si se despeja la Y del primer término, se obtiene:

$$Y = \frac{Y1 - Y0}{X1 - X0} * (X - X0) + Y0$$

El objetivo es conseguir mover los objetos 3D en Tiempo Real

Simplificando para obtener la ecuación (1) y así poder conseguir los datos que se necesitan, se llega a la siguiente expresión:

$$Y = \frac{Y1 - Y0}{X1 - X0} * X + \frac{Y1 - Y0}{X1 - X0} * (- X0) + Y0$$



Figua 2: Aproximaciones a la recta real.

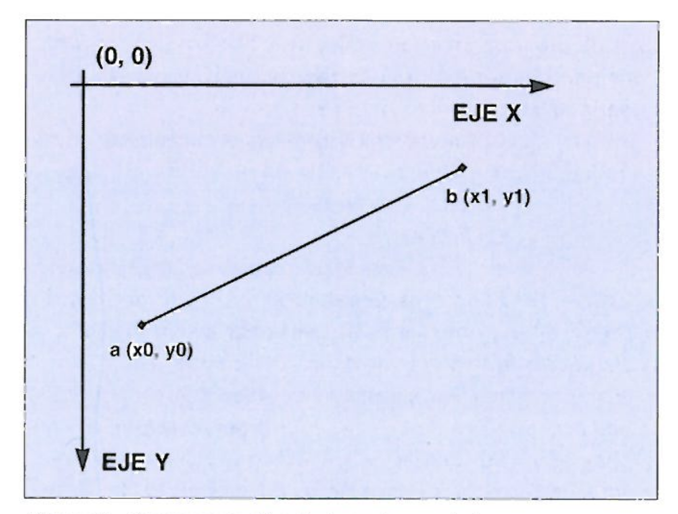


Figura 3: Representación de la recta en el plano.

Por tanto, ya sólo queda asignar a cada variable su parte de fórmula asociada:

$$Pendiente = \frac{Y1 - Y0}{X1 - X0}$$

$$A = Y0 - \frac{Y1 - Y0}{X1 - X0} * X0$$

El algoritmo para lo implementa, se encuentra en el listado 2.

El principal inconveniente de este método radica en la lentitud del bucle principal, ya que hay que realizar una multiplicación por cada pixel que se pinte.

Para evitar ese exceso de cálculo se suelen utilizar algoritmos incrementales, esto es, que guardan los incrementos para pasar de un pixel de la recta al siguiente con lo cual, en el bucle principal se realiza una suma en vez de una multiplicación. El más conocido es el algoritmo de Bresenham, que va a ser el que se explique a continuación.

LISTADO 2

```
Linea_1 (X0,Y0,X1,Y1,Color)
{
        X0,Y0               //Extremo a de la recta
        X1,Y1               //Extremo b de la recta
        Color               //Color de la recta

        Si (X0 = X1)        //Linea Vertical,
                            Pendiente = Infinito
        {
                Si (Y0 > Y1)        //La linea se recorre
                                    de arriba a abajo
                {
                        Intercambiar (Y0,Y1)
                }
                Desde Y = Y0 hasta Y1
                {
                        PintarPixel (X0,Y,Color)
                        Incrementar Y
                }
                Fin_Algoritmo       //Salir del
                                    procedimiento
        }
        Si (X0 > X1)        //La linea se recorre de
                            izquierda a derecha
        {
                Intercambiar (X0,Y0)
                Intercambiar (X1,Y1)
        }
        Pendiente = (Y1 - Y0) / (X1 - X0)
        A = Y0 - Pendiente * X0
        Desde X = X0 hasta X1       //Bucle principal
        {
                Y = Pendiente * X + A
                PintarPixel (X,Y,Color)
                Incrementar X
        }
}                           //Fin Linea_1

Intercambiar (Valor_1,Valor_2)
{
        Auxiliar            //Variable intermedia para
                            intercambiar los valores

        Auxiliar = Valor_1
        Valor_1 = Valor_2
        Valor_2 = Auxiliar
}
```

En primer lugar se deben definir las variables que indican los incrementos globales:

Incremento_X = X1 - X0
Incremento_Y = Y1 - Y0

A continuación, para simplificar la idea, se puede

LISTADO 3

```
Linea_primer_octante (X0,Y0,X1,Y1,Color)
{
        X0,Y0                   //Extremo a de la recta
        X1,Y1                   //Extremo b de la recta
        Color                   //Color de la recta

        Incremento_X = X1 - X0
        Incremento_Y = Y1 - Y0
        Si Incremento_Y = 0             //Linea horizontal
        {
                Diferencial_Y = 0
        }
        Si no                   //Incluye también linea diagonal
            {
            Diferencial_Y = Incremento_Y / Incremento_X
            }
        Y = Y0
        Desde X = X0 hasta X1           //Bucle principal
        {
                PintarPixel (X,parte_entera(Y),Color)
                Incrementar X
                Y = Y + Diferencial_Y
        }
}                               //Fin Linea_Primer_Octante
```

LISTADO 4

```
Linea_2 (X0,Y0,X1,Y1,Color)
{
        X0,Y0                   //Extremo a de la recta
        X1,Y1                   //Extremo b de la recta
        Color                   //Color de la recta

        Si (Y0 > Y1)            //Se coloca en (X0,Y0) el
                                extremo más alto
        {
                Intercambiar (X0,X1)
                Intercambiar (Y0,Y1)
        }
        Incremento_X = X1 - X0
        Incremento_Y = Y1 - Y0
        Si (Incremento_X = 0)           //Linea vertical
        {
                Desde Y = Y0 hasta Y1
                {
                        PintarPixel (X0,Y,Color)
                        Incrementar Y
                }
                Fin_algoritmo           //Salir del
                                        procedimiento
        }
        Si (Incremento_Y = 0)           //Linea horizontal
        {
                Si (X0 > X1)            //Se coloca en (X0,Y0)
                                        el extremo más a
                                        la izquierda
                {
                        Intercambiar (X0,X1)
                        Intercambiar (Y0,Y1)
                }
                Desde X = X0 hasta X1
                {
                        PintarPixel (X,Y0,Color)
                        Incrementar X
                }
                Fin_algoritmo           //Salir del
                                        procedimiento
        }
        Si (Valor_Absoluto (Incremento_X) < Incremento_Y)
        {
            Diferencial_X = Incremento_X / Incremento_Y
            X = X0
            Desde Y = Y0 hasta Y1
            {
                    PintarPixel (parte_entera(X),Y,Color)
                    Incrementar Y
                    X = X + Diferencial_X
            }
            Fin_algoritmo           //Salir del
                                    procedimiento
        }
        Diferencial_Y = Incremento_Y / Incremento_X
        Y = Y0
        Si (X0 > X1)            //Se coloca en (X0,Y0) el extremo
                                más a la izquierda
            {
                Intercambiar (X0,X1)
                Intercambiar (Y0,Y1)
            }
        Desde X = X0 hasta X1
        {
                PintarPixel (X,parte_entera(Y),Color)
                Incrementar X
                Y = Y + Diferencial_Y
        }
}                               //Fin Linea_2
```

considerar un caso básico, cuando ambos incrementos son positivos, *Incremento_X* es distinto de cero y ademas es mayor que *Incremento_Y* (ver figura 4), que es lo mismo que decir que la recta, si se traslada en el plano situando su extremo izquierdo en el origen de coordenadas, se encuentra en el primer octante.

En primer lugar hay que considerar los dos casos básicos:

1.- Incremento_Y = 0. Linea horizontal.

Para dibujar la línea tan sólo sería necesario incrementar la coordenada X de uno en uno desde el punto (X0,Y0) hasta el punto (X1,Y1), sin tocar la coordenada Y.

2.- Incremento_X = Incremento_Y. Linea diagonal.
En este caso habría que incrementar de uno en uno los valores de ambas coordenadas desde el punto (X0,Y0), hasta el punto(X1,Y1).

En el caso general para el primer octante, como el *Incremento_X* es mayor que el *Incremento_Y* lo que se hace es hallar lo que hay que sumar a la coordenada *Y* por cada unidad que se añade a la coordenada *X* y al valor obtenido se le llama *Diferencial_Y*. Como el número obtenido es decimal, al procedimiento *pintar_pixel* se le llama con la parte entera de *Y* ya que las coordenadas de pantalla deben ser enteras. El algoritmo resultante está en el listado 3.

Un simple vistazo al algoritmo basta para darse cuenta de que es mucho más rápido que el anterior gracias al ahorro de la multiplicación por cada pixel. Ahora hay que extrapolar la idea al resto de las posibles inclinaciones de la recta.

Lo primero es situar el punto más alto (el de menor coordenada *Y*) como primer extremo para pintar la línea de arriba a abajo. Se podría haber tomado otro sentido, pero como éste va a ser el que se utilice en el algoritmo de pintar polígonos, es mejor hacerlo así para no tener que variar más adelante.

El algoritmo de Cohen-Sutherland es el método más rápido

El siguiente paso es considerar las líneas horizonzales y verticales como casos particulares y por último dividir los casos que nos queden en función de la pendiente de la recta. La división quedaría de la siguiente forma:

Caso 1.- Linea horizontal.
Caso 2.- Linea vertical.
Caso 3.- -1 < Pendiente < 1
Caso 4.- Resto.

La diferencia entre los casos 3 y 4 radica en que en el bucle principal uno recorrería el eje X sumando *Diferencial_Y* a la coordenada Y (igual que en el caso inicial), mientras el otro recorrería el eje Y sumando en este caso *Diferencial_X* a la coordenada X. El algoritmo se encuentra en el listado 4.

Este va a ser el algoritmo que se incluya en la librería. Ha sido implementado en el lenguaje ensamblador para lograr mayor velocidad.

El procedimiento que pinta líneas en pantalla es el siguiente:

void linea (X0,Y0,X1,Y1,Color,página_oculta)

Para que el procedimiento funcione los puntos que definen la recta deben estar situados dentro de la ventana de visión, ya que no se puede pintar un punto en una posición que no existe. Para solventar el problema hay que añadir un nuevo procedimiento que recorte la línea previamente para que no haya problemas al pintarla.

RECORTE DE RECTAS CON LA VENTANA DE VISION

Existen numerosos métodos para recortar las líneas. El más simple es detectar si el punto que se va a pintar cae dentro de pantalla, pero es poco eficiente ya que hay que comprobar todos los puntos estén o no en pantalla.

El método más rápido consiste en hallar los puntos de corte, si los hubiera, con la pantalla de visión. Uno de los algoritmos más conocidos para realizar dicha operación es el de Cohen-Sutherland, que además destaca por su simplicidad y eficiencia.

El algoritmo divide el plano en nueve regiones (ver figura 5), y asigna a cada región un número compuesto por cuatro bits, o bien si se quiere implementar de forma cómoda desde C, una estructura compuesta por cuatro números que pueden tener el valor 0 ó 1. Cada bit indica una posición en el plano y está a 1 si está situado en esa posición y 0 en caso contrario.

Para recortar la recta, se comprueba la posición en el plano de los extremos, asignándoles el número asociado a su posición.

A continuación se hace un OR entre los números de los extremos, en caso de obtener como resultado 0, ambos puntos están dentro de la pantalla y por tanto no hay que recortar la recta.

Después se hace un AND con los dos números y si se obtiene un número distinto de cero, implica que ambos puntos están en el mismo lado del plano, por tanto la recta está fuera de la pantalla, y no hay que dibujarla.

Si no se cumple ninguno de los casos anteriores, hay que coger el punto situado fuera de la pantalla (aquél

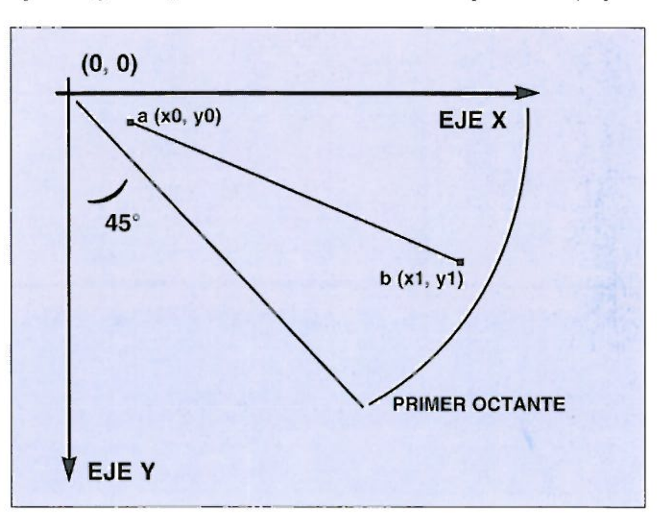


Figura 4: Recta situada en el primer octante.

que tenga un número asociado distinto de cero), hallar el punto de corte de la recta con el borde de la pantalla por donde se sale y sustituirlo por el punto que se ha obtenido mediante dicho corte.

Por último, se recalculan los números asociados a cada punto y se repite la operación hasta que ambos puntos están dentro de la pantalla.

El procedimiento *linea()* incluido en la librería, se encarga de llamar a la rutina de recorte para evitar tener que realizar dos llamadas.

DAC DE VIDEO

La paleta de colores por defecto de la VGA está bastante descompensada, y si se busca un color concreto con una tonalidad concreta, las probabilidades de acertar son reducidas, pero si además se quiere que un objeto al alejarse varíe su tonalidad volviendose más oscura, para acentuar la sensación de tridimensionalidad, la situación puede convertirse en desesperante.

La paleta de colores de la VGA tiene un *popurrí* de colores que conviene adaptar en el programa

Por ello, se ha incluido una paleta a medida en la librería. El procedimiento *svmode()* se encarga de situarla en lugar de la que existe por defecto.

La ventaja consiste en que está dividida en dieciséis bancos de dieciséis tonalidades cada uno. Por tanto, si se elige el color en hexadecimal el primer carácter indica el color propiamente dicho y el segundo su tonalidad, que variará desde "0" la más oscura hasta "F" la más clara.

Por ejemplo, si se tiene el color "7Ah" y se quiere oscurecer, el nuevo color será el "79h", en cambio si se quiere aclarar, el color resultante sería el "7Bh".

Se admiten todo tipo de sugerencias para elegir una paleta a gusto de todos. Para ello lo mejor es realizar la petición de colores por carta a la revista indicando los dieciséis que se consideren más importantes, por ejemplo: cyan, índigo, verde esmeralda,...

Se intentará satisfacer a todo el mundo eligiendo una paleta acorde con los colores más solicitados o en caso de que exista una gran variedad se podría intentar idear un sistema cómodo para que cada cual eligiese su paleta.

CONCLUSION

Por fin se puede conseguir el tan ansiado movimiento fluido, libre de molestos parpadeos, pero aún siguen existiendo pegas, que poco a poco irán desapareciendo. Una de las más importantes es la forma de crear objetos. Definirse todos los puntos puede llegar a ser eterno, además la mayoría de los puntos se calculan al menos dos veces con la consiguiente disminución de velocidad, por ello, se ha empezado a crear un editor de objetos para simplificar la construcción. También se va a modificar la rutina de calcular rectas para evitar la repetición de cálculos. Pero todo esto lleva su tiempo. De todos modos, las mejoras se irán incluyendo en el programa que acompaña al artículo a medida que se vayan terminando.

EL MES QUE VIENE

Como lo prometido es deuda, para el siguiente artículo se realizará una demostración de las posibilidades que ofrecen las rutinas que se han visto. Pronto se empezarán a construir los objetos mediante polígonos y llegados a ese punto comenzarán a explicarse las técnicas de ocultación de caras, para que aparezcan en pantalla sólo las caras visibles, pero eso será a partir del mes que viene. ■

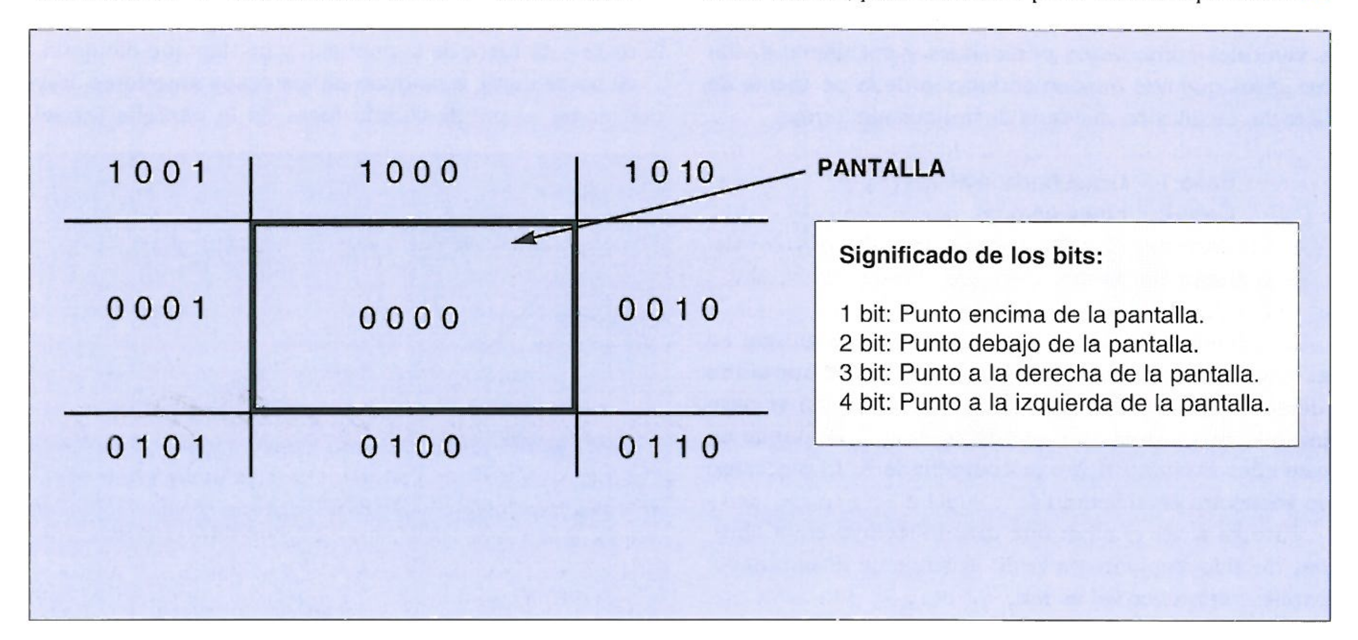


Figura 5: División del plano según el algoritmo de *Coher-Sutherland*.

DIRECTIVAS DEL LENGUAJE

Ricardo L. Cano

En este artículo se continuarán explicando las instrucciones del procesador y las directivas más importantes del lenguaje ensamblador. Al final del mismo se analizarán dos versiones, una correcta y otra errónea, de un mismo programa.

REPERTORIO DE INSTRUCCIONES

Este doble análisis servirá por un lado para comprobar la aplicación práctica de las instrucciones y directivas ya estudiadas, y por otro para comenzar a acostumbrar al lector a rehuir los tipos de errores más generales y molestos de la programación en ensamblador.

Como se dijo en el artículo anterior, las instrucciones se dividen en diversas categorías. En el capítulo precedente se estudiaron las instrucciones más importantes de transferencia de datos y de operaciones aritméticas. Hoy se comenzará con el estudio de las que se encargan del control de la ejecución del programa. La lista completa de las mismas puede verse, al final del presente artículo, incluida en la tabla de referencia de las instrucciones del procesador. En esta lista la explicación de cada instrucción está necesariamente resumida por lo que la tabla sólo debe ser utilizada para consultas generales.

INSTRUCCIONES DE CONTROL DE EJECUCION DEL PROGRAMA

Dentro de este grupo de instrucciones, la más simple es JMP (JuMP=saltar) que provoca un salto incondicional a la dirección especificada por el operando que acompaña a la instrucción. Esta bifurcación puede caer dentro del mismo segmento donde se halla la instrucción de salto o en otro diferente. En el primer caso sólo es alterado el registro IP y en el segundo también queda modificado el registro del segmento de código CS. Además el salto puede ser directo, si se especifica una etiqueta, o indirecto, si el operando empleado es un registro o una dirección de memoria. En el primer caso al registro IP se le suma el desplazamiento contenido en la propia instrucción de salto y en el segundo el valor de IP se sustituye por el del operando. Pueden verse algunos ejemplos de saltos en el listado 1.

El siguiente tipo de salto, el condicional, precisa del cumplimiento de una condición dada para que la bifurcación llegue a producirse. Esta condición dependerá de cual sea el salto condicional escogido; JC pregunta por el flag de carry, JZ por el flag ZF, JS por el de signo, etc. Si la condición no se cumple el programa continuará con la instrucción que sigue a la del salto. Otra diferencia con respecto a JMP es que el rango de offset es bastante pequeño; el programador solo dispone de 128 bytes para las posiciones que preceden al mnemónico del salto y 127 para las posteriores. Esta restric-

ción se debe a que el ensamblador solo utiliza un byte para indicar el desplazamiento en el salto condicional (pero a partir del 386 esta restricción desaparece). Esto es bastante molesto, pero el programador deberá cuidar de que la etiqueta a la que se pretenda bifurcar quede dentro de este rango de desplazamiento. De lo contrario el ensamblador indicará el error correspondiente.

El funcionamiento de CALL (llamar) ya ha sido explicado anteriormente, junto al de la pila. CALL es utilizada para efectuar las llamadas a los procedimientos. Un procedimiento es un bloque de instrucciones creado por el programador para realizar una tarea concreta, o sea una rutina o bien una subrutina. Una vez que concluye la tarea prevista, cuando se encuentre una instrucción RET (retorno), se voverá a la línea que sigue a la CALL de llamada. El propósito de los procedimientos es suministrar al programador de ensamblador un mecanismo que le permita escribir programas de manera estructurada y eficiente.

LISTADO 1

```
INSTRUCCIONES JMP VALIDAS.
JMP rutina1     ;saltar a la etiqueta rutina1

MOV BX,offset rutina1
JMP BX          ;saltar a la posicion dada por el desplazamien-
to
                ;en BX. O sea a CS:BX (a rutina1)

rut1jmp dw rutina1 ;(esto está en el segmento de datos)
JMP rut1jmp     ;saltar a la direccion guardada en rut1jmp (o sea
                ;a rutina1)
```

LISTADO 2

DIRECTIVAS PARA LA DEFINICION DE DATOS

Directiva	Funcion	Sintaxis y Valor maximo
DB	Definir byte	variable DB 255
DW	Definir palabra	variable DW 65535
DD	Definir doble palabra	variable DD 4294967295
DQ	Definir cuadruple palabra	variable DQ 18446744073709551615
DT	Reservar diezbytes de memoria	variable DT 1208925819614629174706176

ALGUNAS DIRECTIVAS

Aunque aun quedan por estudiar muchas instrucciones y también examinar con más detalle algunas de las ya explicadas, conviene examinar unas cuantas directivas a fin de comprender mejor la plantilla y el programa de ejemplo suministrados en el último apartado. Como se dijo en el primer capítulo, las directivas figuran como parte del listado del programa y son órdenes para el ensamblador, a fin de que éste, al compilar el programa, sepa cómo ha de interpretar datos e instrucciones. Las directivas más importantes son las de definición de datos, las de referencias externas y las de definición de segmentos y procedimientos. Las primeras se utilizan para crear las variables del programa y reservar la memoria que necesitan. Las de referencias externas se emplean en programas de mediano o gran tamaño, cuando se tienen varios módulos y se desea que uno conozca las variables y datos de otros. Y las de segmentos y procedimientos nos permiten especificar el principio y final de un segmento o una estructura. Para la definición de datos, el lector puede examinar el listado 2 donde se muestran las directivas que intervienen en la creación de variables y algunos ejemplos de las mismas. En todas estas directivas el formato es;

```
nombre_variable DIRECTIVA lista_de_valores. . .
```

El primer campo es opcional y es el nombre asignado al primer byte o a la primera palabra, etc. La memoria que se reserva depende de si la directiva es un DB o un DW u otra. En cuanto a la lista de valores puede consistir en constantes numéricas positivas o negativas, separadas por comas y cuyo valor ha de quedar dentro del rango permitido por la directiva. Si se ha empleado DB los valores no excederán 255 (o -128 y +127), si se ha usado DW no se pasará de 65535 (o -32768 y +32767), etc. Asimismo puede utilizarse el carácter "?" que indica que se reserva un elemento de memoria, pero que su valor inicial es indefinido. Aquí puede intervenir el operando DUP para indicar una repetición de n veces de él o los elementos de memoria, tal y como se aprecia en los ejemplos que siguen;

```
variable1 DB 40 DUP(?)
pila  DB 128 DUP('memoria')
```

Variable1 reserva 40 bytes de contenido indefinido mientras que en pila se reservarán 128*7 bytes (la palabra memoria, repetida 128 veces).

También, aunque sólo para el caso de DB, puede crearse una cadena de caracteres delimitada por comillas simples o dobles, donde cada carácter equivale a un byte. Y finalmente DW puede utilizarse para definir y guardar el desplazamiento de otra variable mientras que DD puede

LISTADO 3

```
Ejemplos de referencia de variables
;Supóngase que en el segmento de datos tenemos
las siguientes definiciones;
...
variable1            DB 210,212,213
variable2            DW 5000
direcc1              DW variable2
direcc2              DD variable2
...
;En el segmento de código serían validas las
siguientes instrucciones;
...
MOV AL,variable1     ;AL=210
MOV CX,variable2     ;CX=5000
MOV BX,direcc1       ;BX=offset a la variable2
MOV AX,[BX]          ;AX=5000 (ver direc. indirecto
                     por registro)
ADD [BX],CX          ;variable2=10000
LDS SI,direcc2   ;SI=offset de variable2 y DS=segmento de la
;misma. LDS es una instrucción de transferencia que
se tratara mas adelante.
...
```

hacer lo propio con una dirección completa de memoria con su segmento y offset. Ahora véanse algunos ejemplos acerca de cómo podrían referenciarse un grupo de variables desde una zona de código en el listado 3.

Otra directiva de importancia es EQU que asigna un valor a un nombre simbólico. Su formato es;

nombre EQU expresión

Donde la expresión puede ser una constante numérica, una referencia de dirección, otro nombre simbólico o cualquier combinación de expresiones que pueda traducirse como un valor. Esta utilidad sirve sobre todo para hacer más claro el fuente del programa y para hacer más sencillas las alteraciones ya que será más breve cambiar la expresión asignada al nombre que cambiar todas las instrucciones donde debiera ir el valor de la expresión, si no se utiliza EQU. El valor de la expresión no puede redefinirse a lo largo del fuente. Dos posibles ejemplos son;

```
pantalla EQU 0A000h
filas EQU 25
columnas EQU 80
```

Donde el primer EQU asigna a la constante pantalla la dirección del segmento adecuado para hacer operaciones en la memoria de vídeo. El segundo da a la constante filas el valor 25 y el tercero el valor 80 a columnas.

En cuanto a las directivas de segmentos y procedimientos, PROC y ENDP marcan el principio y el fin de un procedimiento. Este puede ser NEAR (cercano) o FAR (lejano) dependiendo de si va a ser llamado desde el mismo segmento o desde otro distinto al de llamada. Por ello, lógicamente, al llamar (CALL) a un procedimiento NEAR sólo se guarda el desplazamiento IP de la instrucción siguiente en la pila (ver apartado correspondiente sobre el stack en el capítulo anterior), mientras que en el caso de un FAR se guardará también el segmento CS. El formato de un procedimiento cualquiera puede verse en el listado 4.

En el listado nombre_procedimiento es el nombre de la etiqueta que será objeto de la llamada CALL o de la instrucción de salto. Nótese que las directivas PROC y ENDP no son obligatorias. Sin ellas se entiende que cuando la etiqueta lleva el carácter ":" al final del nombre, es un procedimiento NEAR y FAR en caso contrario.

Por otro lado, el primer procedimiento en ejecutarse queda especificado por la directiva END que va acompañada por la dirección de comienzo del programa. Esta dirección normalmente es una etiqueta que corresponde al nombre del primer procedimiento. END indica además el final del programa fuente.

En cuanto a los segmentos, como ya se explicó en el primer capítulo, existen cuatro tipos; el de código referenciado por el registro CS, el de datos indicado por DS, el extra direccionado por ES y el de la pila apuntado por SS. Las directivas SEGMENT Y ENDS son las que marcan el principio y el fin de estos segmentos con el siguiente formato:

```
nombre_segmento SEGMENT parámetros
 . . .
nombre_segmento ENDS
```

El interior del segmento puede corresponder a instrucciones o definiciones de variables según el tipo de segmento definido. Los parámetros sirven para dar diversas instrucciones al programa de linkado y no tienen demasiada importancia, por el momento.

Las directivas de segmentos, PROC y ENP, marcan el inicio y conclusión de un procedimiento

Finalmente, para asignar a cada segmento creado un registro adecuado contamos con la directiva ASSUME. Indica al ensamblador que registro de segmento se asigna para direccionar cada segmento definido con SEGMENT. Naturalmente se utilizará el CS para el segmento de código, DS para el de datos y SS para el de pila.

Ahora el lector puede examinar la plantilla de programación en el fichero FIG5.TXT del disco. Esta plantilla se suministra como un fichero de texto para que el usuario escriba sus propios programas en ensamblador. La plantilla podría ser diferente (sobre todo si hay

varios módulos de ensamblador en el proyecto o si el módulo de ensamblador está "enganchado" a un programa en otro lenguaje), pero de momento servirá a los fines previstos.

LOS PROGRAMAS DE EJEMPLO

El listado correspondiente al programa de ejemplo de este mes, puede verse en el fichero FIG6.TXT. Seguidamente se comentará cada línea para refrescar los conceptos del lector. Además, simultáneamente se examinará otro ejemplo del mismo programa (en el archivo FIG7.TXT), pero con la particularidad de que dicho ejemplo está lleno de errores de todo tipo introducidos adrede a fin de ilustrar mejor los efectos de los diversos fallos que pueden cometerse. Naturalmente pueden darse muchos más "bugs" de los que aparecen en el ejemplo y por ello el programador novato debe estar preparado psicológicamente para entregar un fuente al compilador de ensamblador y recibir a cambio un largo listado de errores, en vez de un programa ejecutable. Contra esto sólo puede recomendarse una cosa: paciencia.

El propósito del programa de ejemplo es sencillo. Simplemente se pretende mostrar el funcionamiento de las instrucciones y modos de direccionamiento más comunes. Para hacer más evidentes los resultados del programa, los movimientos de datos se efectúan en el segmento donde comienza la memoria de la VGA, después de haber activado el modo gráfico de 320*200 pixels con 256 colores. De esta manera los resultados de los posibles cambios que decida introducir el lector podrán apreciarse más fácilmente.

Como puede verse en el listado correcto, hay cuatro procedimientos de los cuales uno es de tipo FAR mientras que los restantes son NEAR. Aquí conviene recordar que los procedimientos NEAR sólo pueden ser llamados por una instrucción que se halle en el mismo segmento que estos. Dado que no existe forma de saber desde donde va a ser llamado el programa resulta claro entonces que el primer procedimiento en ejecutarse debe ser siempre de tipo FAR, ya que éstos si pueden ser llamados desde otros segmentos (recuérdese que esto es así porque al hacer la llamada a un procedimiento FAR se guarda el registro de segmento además del de desplazamiento). Lógicamente también la etiqueta que acompaña a la directiva END corresponde a un procedimiento FAR. En el ejemplo correcto este procedimiento FAR es el PRUEBA. Aquí pueden verse ya varios errores en el listado incorrecto. En primer lugar todos los procedimientos son NEAR, en segundo ha sido olvidada la colocación de la marca de fin de procedimiento ENDP, de PRUEBA y por último la etiqueta que sigue a END corresponde a una rutina local, que para colmo (como se verá después) tiene descompensada la pila por lo que no se puede retornar normalmente desde ella.

LISTADO 4

```
Formato de un procedimiento.
nombre_procedimiento    PROC    atributo (NEAR o FAR)
            lista de instrucciones
            ...
            RET
nombre_procedimiento    ENDP        ;marca de fin de
                                    procedimiento
```

En el programa se definen tres segmentos; el de datos donde solo se ha reservado espacio para dos variables (pantalla y dirpant), el de pila con espacio para 128 bytes (muchos más de los que necesita la pila de este programa) y el segmento de código.

Las dos primeras instrucciones que se ejecutarán se encargan de meter en el registro DS la dirección del segmento de datos que ha sido definido (recuérdese que no puede meterse un valor inmediato directamente en un registro de segmento. Hay que utilizar un registro).

El modo gráfico escogido para el ejemplo es el 320x200 puntos con 256 colores

Seguidamente vienen tres llamadas CALL a otros tantos procedimientos. El primero (ponmodo) activa el modo gráfico deseado con una llamada a la BIOS hecha mediante la instrucción INT. Esta instrucción, junto con diversas entradas del BIOS y del DOS, será estudiada en próximos capítulos. Por el momento basta con que el lector sepa que esta rutina activa el modo gráfico más corriente de las VGA, gracias al valor que se introduce en el registro AX y que sirve como parámetro para la función invocada por la interrupción.

Aquí puede advertirse otro error ya que en el listado incorrecto ha sido olvidada la "h" que indica que el valor que se guarda en AX es un número hexadecimal por lo que el número se interpreta como el 13 decimal mientras que el 13 hexadecimal equivale al 19 decimal.

El siguiente procedimiento (ciclopan) provoca un rápido ciclo de cambio de colores. El sistema consiste en cambiar rápidamente todo el contenido de la memoria RAM de vídeo. En el modo escogido de 320*200 puntos con 256 colores, la extensión de la misma es de 64000 bytes o sea de casi un segmento completo. Para poder acceder a ella, ciclopan mete en DS la dirección donde comienza el segmento de la memoria de vídeo. Esta dirección seria linealmente la 655360 pero dado que las direcciones se forman con dos registros bastará con dividir este número por 16 para obtener el valor necesario para el segmento (40960 en decimal o 0A000h en hexadecimal). Así, si el registro de desplazamiento vale 0 se está direccionando el primer byte de la memo-

ria de pantalla, con un offset de 1 se apuntará al segundo byte y así sucesivamente. Dicho esto resulta claro que después de haber puesto el valor 0A000h en DS bastará con manipular un registro de desplazamiento para poder manipular toda la memoria de vídeo.

En el modo 13h, cada byte corresponde a un pixel. Ciclopan llena toda esta área de memoria con un valor que indica la posición del color en la paleta y luego incrementa dicho valor para referenciar el siguiente color de la paleta y vuelve a llenar la pantalla con este. El bucle se repite tantas veces como colores permite la paleta (256) por lo que puede verse una rápida oscilación de colores en el monitor. Al concluir el ciclo se retorna al procedimiento principal con RET. El funcionamiento de esta rutina (y el de todo el programa) podía haberse optimizado mucho pero era preciso utilizar, en lo posible, solo las instrucciones ya explicadas.

Examinando la rutina con algo más de detalle se advierte como BX pasa a cumplir el papel de registro de offset. Es puesto a cero antes de comenzar el bucle y va incrementándose (INC) dentro de éste. Los valores de color van siendo insertados en memoria empleando el direccionamiento relativo a base (MOV [BX],AL) y el registro CX es utilizado como contador de los 64000 puntos de pantalla que hay que llenar con el valor de color que guarda AL.

Después de cada operación de decremento sobre CX, la siguiente instrucción es un salto condicional (JNZ bucle2) que salta a la etiqueta adjunta siempre que CX aun no haya llegado a cero. Cuando esto sucede, el flag de cero del registro de banderas se alza y la condición exigida deja de cumplirse por lo que el curso del programa abandona este bucle y se incrementa AL para reiniciar el bucle de llenado de memoria con el siguiente color que haya en la paleta. Sólo cuando el registro DL, empleado aquí como contador del número de colores a volcar (los 256 de la paleta), llega a cero, concluye la rutina. Se recupera con POP el valor de DS que había sido guardado en la pila (con PUSH) y que apuntaba al segmento de datos del programa antes de entrar en ciclopan, y se retorna.

Hasta aquí lo referente a la versión correcta. En la versión defectuosa de ciclopan la pila queda descompensada ya que el PUSH del principio queda anulado por culpa de un punto y coma que convierte la instrucción en un simple comentario. ¿La razón de esto? Un posible despiste pero como al final de la rutina POP "recupera" el valor que no ha sido guardado por el PUSH anulado, sucede en realidad que POP toma la dirección que debía servir a RET para retornar, con lo que al llegar a esta instrucción la ejecución del programa marcha a algún punto impredecible produciéndose un seguro "cuelgue" del mismo. Otro error importante es el olvido de los corchetes donde debía guardarse el valor de AL en el desplazamiento indicado por BX.

Al no haber corchetes el direccionamiento relativo a base se convertiría en direccionamiento por registros pero como los registros implicados no tienen la misma longitud en bits hay un error de sintaxis que será indicado durante la compilación.

En cuanto al procedimiento *ciclolin* es muy parecido a la rutina descrita, pero esta vez el bucle de color va cambiando en cada línea lo que da una falsa impresión de scroll vertical. Una de las pocas diferencias es que el valor del segmento de pantalla es introducido en DS de una manera algo más retorcida a fin de mostrar al lector que la variable dirpant guarda la dirección a la variable pantalla. Naturalmente el valor 0A000h podía haber sido cargado directamente en AX y después en DS. En cuanto a la versión defectuosa de *ciclolin*, el fallo principal está en que el registro de segmento DS no es puesto con la dirección del segmento de la RAM de vídeo.

En su lugar el registro de segmento ES toma este valor pero como la instrucción MOV [BX],AL actúa sobre el segmento de datos y no sobre el ES, lo que ocurre es que el llenado de los 64000 bytes de color se efectuaría sobre el segmento de datos, estropeando todos lo que allí hubiera. Además aquí también hay otro caso de pila descompensada, por lo que tampoco habría un retorno correcto al procedimiento prueba. En el resto del listado hay muchos otros ejemplos de posibles fallos; valores inmediatos que exceden la capacidad de los registros donde deben almacenarse, contadores incorrectos, bucles mal puestos, etc.

LOS ERRORES MAS COMUNES

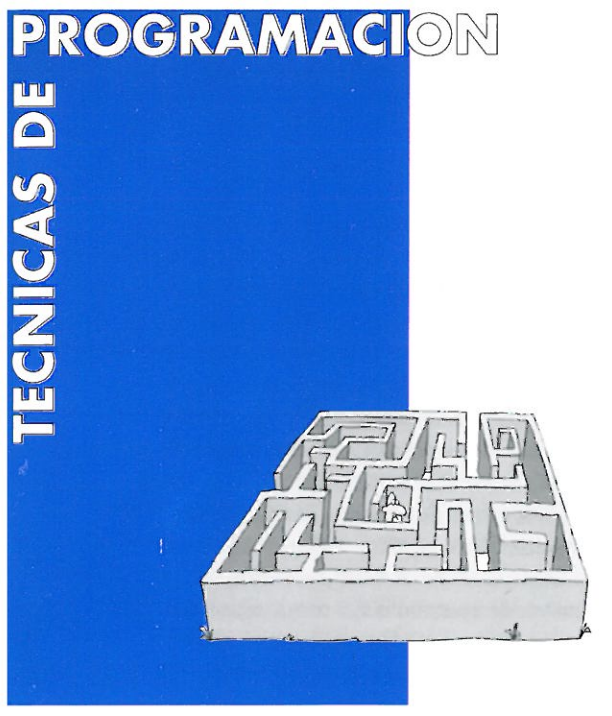
Un examen atento de los errores "cometidos" en el fichero FIG7.TXT, puede ahorrar algunos problemas al lector. Quizá uno de los fallos más comunes sean los que se producen con los registros de segmento, sobre todo con el DS. Esto es así porque como se ha visto las instrucciones de movimiento de datos utilizan los segmentos y si estos no apuntan a donde deben, no se estarán realizando las transferencias correctamente aunque el compilador no indique ningún mensaje de error. En el listado de fallos cada bug ha sido indicado con los comentarios correspondientes. Estos errores pueden causar efectos diversos siendo el más corriente el "cuelgue" o "crack" del ordenador. En dicho caso no quedará otra alternativa que apretar el botón de reset .

El error más corriente en estos casos es el "cuelgue" o "crack" del sistema operativo del ordenador

PRUEBAS

Si se dispone por ejemplo del ensamblador de Microsoft, el programa puede ser compilado y linkado con las ordenes;

```
masm prog2. asm
link prog2. obj
```



METODOS DE ORDENACION

David Rubio

El tema que presentamos este mes es muy interesante por dos motivos principalmente. El primero de ellos, y más evidente, es la necesidad de ordenar una serie de datos siguiendo un criterio determinado. El segundo es mostrar la diversidad de algoritmos que se pueden emplear para realizar la misma operación, viendo cuáles de ellos son más óptimos, qué ventajas tienen unos frente a otros. Una vez más se hará hincapié en uno de los objetivos de esta sección: no hay que desarrollar un algoritmo que funcione, hay que buscar el que lo haga de la mejor forma posible.

Los métodos de ordenación se pueden dividir, primeramente, en dos tipos:

- ordenación interna. La que tiene lugar en la memoria del ordenador.
- ordenación externa. Se realiza sobre elementos almacenados externamente, como pueden ser discos duros, disquetes, etc.

ORDENACION DE VECTORES

En este artículo se verán únicamente los métodos de ordenación interna. Estos los aplicaremos sobre vectores (ARRAYS) de elementos. En el lenguaje de programación usado la declaración de los datos a ordenar es la siguiente:

```
CONST
  N = ?;

TYPE
  tElemento = Integer;
  tIndice = 1..N OF Integer;
  tVector = ARRAY [tIndice] OF tElemento;
```

Se utilizarán dos valoraciones para ver la eficacia de cada método de ordenación. El número de comparaciones de claves, llamado C, y el número de movimientos de la mismas, llamado M. Ambos son función del número n de elementos que componen el vector a ordenar.

Cuando se programa muchas veces es necesario ordenar una serie de datos según algún criterio. En este número se analizan los principales algoritmos de ordenación de datos.

CARACTERISTICAS DE LOS METODOS DE ORDENACION

Existen tres características que se han de tener en cuenta a la hora de estudiarlos. Estas son:

Natural / Antinatural: el método aplicado es natural cuando el vector ya está bastante ordenado. Cuando no lo está se dice que es antinatural. Por ejemplo el vector [1,3,8,10,9] tarda menos en ordenarse que el [8,1,10,9,3]. El primero corresponde a un método natural y el segundo a uno antinatural.

Estabilidad: Si existen claves iguales el método respeta el orden relativo que tenían dichas claves antes de la ordenación.

Complejidad: Es una manera de obtener una función matemática para saber cúantas veces se ha realizdo una operación.

CLASIFICACION

Los métodos de ordenación se dividen en dos tipos:

-Directos. Suelen ser algoritmos cortos y fáciles de entender, aunque algo lentos. El número de comparaciones que requieren es del orden de n2.

De entre todos ellos se estudiarán los siguientes:

- Inserción directa
- Inserción binaria
    - Selección directa
    - Intercambio
        - Burbuja
        - Sacudida

-Avanzados. Mucho más complejos que los anteriores.

Se estudiará el método de ordenación rápida o QuickSort.

De estos dos grandes grupos se verán primeramente los métodos directos por tres razones:

- Utilizan los principios de ordenación más usuales, imprescindibles para conocer la mecánica de cualquier tipo de ordenación.

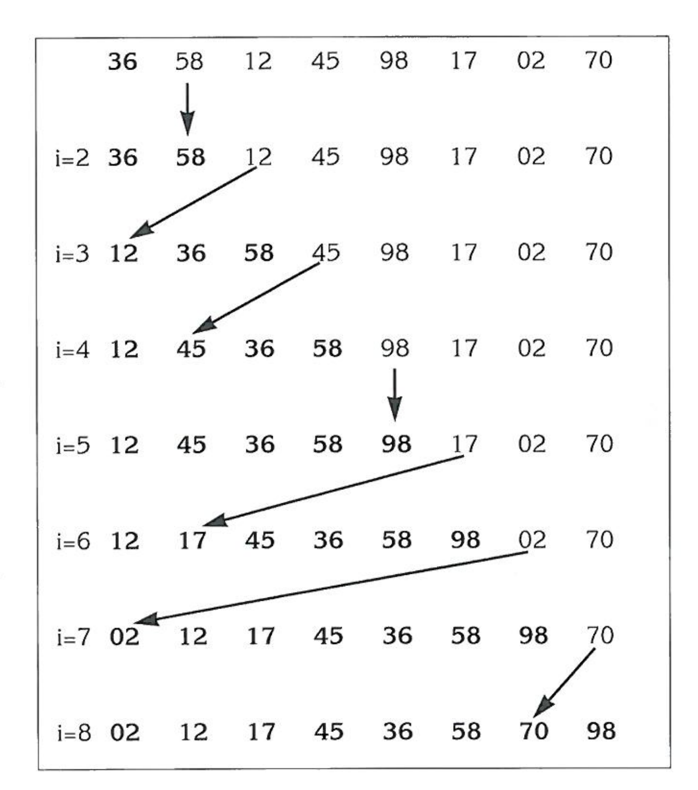
- Son algoritmos cortos y bastante fáciles de entender.

- Para n bastante pequeña se muestran más eficaces que los métodos avanzados, ya que éstos realizan unas operaciones muy complejas que tardan más en ejecutarse que las de los directos.

METODOS DIRECTOS

INSERCION DIRECTA:

Los elementos están divididos en una secuencia destino a1...ai-1 y en una secuencia origen ai...an. En cada paso, empezando por i=2 e incrementando i de uno en uno, se toma el elemento i de la secuencia origen y se transfiere a la secuencia destino insertándolo en el sitio apropiado. Veamos cómo funciona con un ejemplo. Tenemos el vector [36,58,12,45,98,17,02,70]. El siguiente cuadro muestra las etapas de la ordenación para cada i.



El algoritmo en Pascal es el siguiente:

```
PROCEDURE InsercionDirecta;
VAR
  i,j:tIndice;
  Elemento:tElemento;
BEGIN
  FOR i:=2 TO N DO
  BEGIN
    Elemento:=Vector[i];
    Vector[0]:=Elemento;
    j:=i-1;
    WHILE Elemento < Vector[j] DO
    BEGIN
      Vector[j+1]:=Vector[j];
      j:=j-1
    END
    Vector[j+1]:=Elemento
  END
END;
```

Los totales de comparaciones y movimientos de este método son:

$C_{min} = n-1$ $\qquad M_{min} = 2(n-1)$

$C_{med} = 1/4\ (n^2+n-2)$ $\qquad M_{med} = 1/4\ (n^2+9n-10)$

$C_{max} = 1/2\ (n^2+n)-1$ $\qquad M_{max} = 1/2\ (n^2+3n-4)$

Cuando todos los elementos están ordenados desde el principio, el número de comparaciones y de movimientos es el reflejado en las fórmulas anteriores. En este caso se comporta como un método natu-

ral. Y también es estable porque no modifica el orden de los elementos repetidos. Las comparaciones y movimientos máximos los efectúa cuando los elementos del vector están en orden inverso. Se puede decir que el método de inserción directa es del orden de O(n2)

INSERCION BINARIA:

Se puede mejorar el método de inserción directa, de tal forma que a la hora de insertar en la secuencia destino, en vez de hacerlo secuencialmente lo se puede hacer con el método de búsqueda binaria. Este método arranca del elemento central de la secuencia de destino y continúa por bisección hasta encontrar el punto de inserción.

El algoritmo en Pascal es el siguiente.

```
PROCEDURE InsercionBinaria;

VAR
 i,j,iz,de,m:tIndice;
 Elemento:tElemento;

BEGIN
 FOR i:=2 TO N DO
 BEGIN
  Elemento := Vector[i];
  iz:=1;
  de:=i-1;
  WHILE iz <= de DO
  BEGIN
   m:=(iz+de) div 2;
   IF Elemento < Vector[m] THEN
    de:=m-1
   ELSE
    iz:=m+1
  END;
  FOR j:=i-1 DOWNTO iz DO
   Vector[j+1]:=Vector[j];
  Vector[iz]:=Elemento
 END
END;
```

El número de comparaciones es independiente del orden inicial de los elementos. Sin embargo, debido a que la división utilizada en la etapa de bisección trunca el resultado, el número de comparaciones necesarias con i elementos puede aumentar en una sobre las esperadas. Esta circunstancia hace que las posiciones inferiores de inserción se localicen ligeramente más rápido que las superiores. Por tanto esto favorece los casos en los que los elementos están inicialmente muy desordenados. Necesita el mínimo de comparaciones cuando los elementos están en orden inverso inicialmente y el máximo cuando ya están ordenados. Este es un caso de comportamiento antinatural. En cuanto a la estabilidad depende de como se programe; si ponemos <= no es estable, pero si ponemos < sí es estable.

La mejora obtenida en este método se refleja en el número de comparaciones y no en el número de movimientos. El número de comparaciones es del orden de $O(n \cdot log2n)$.

Dado que el movimiento de los elementos es mucho más costoso en tiempo que las comparaciones, la mejora de este método no es ni mucho menos drástica. Además, la reordenación de arrays ya ordenados necesita más tiempo que con la inserción directa. Con este método se ve como lo que en principio es una mejora aparente, se puede convertir en muchos casos en un empeoramiento. Luego no es bueno para programar: no es económica la inserción de un elemento a costa de desplazar una posición todos los elementos de una fila. Podrían esperarse mejores resultados de un método en que los movimientos de elementos se realizaran sobre elementos aislados y con saltos más largos entre posiciones de los mismos. Esta idea nos conduce a la ordenación por selección.

SELECCION DIRECTA:

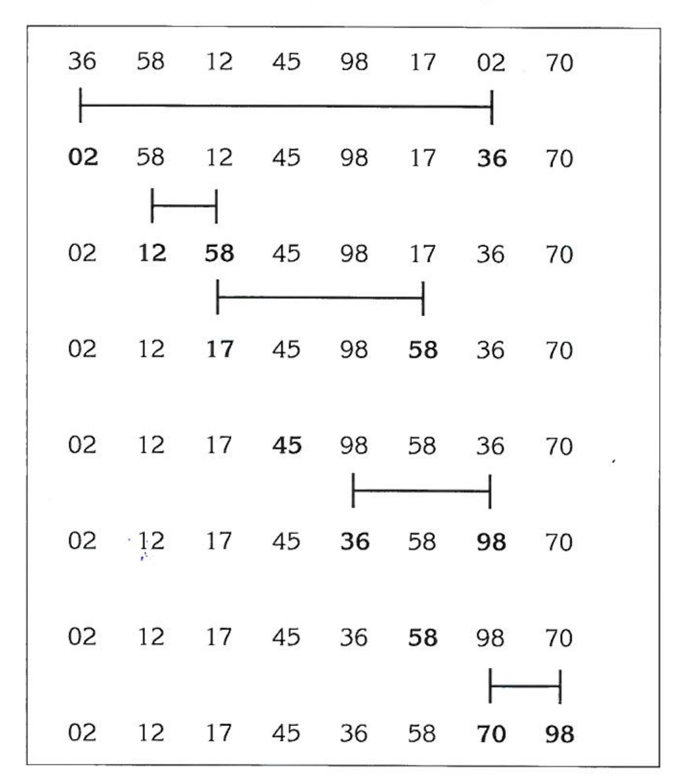
Este método se basa en los siguientes principios:

1.- Seleccionar el artículo con clave mínima.

2.- Intercambiarlo con el primero a1.

3.- A continuación se repiten estas operaciones con los elementos n-1, n-2, restantes, hasta que quede un único elemento, el mayor.

Siguiendo con el ejemplo propuesto anteriormente, la secuencia de ordenación quedaría de la siguiente manera:





El algoritmo correspondiente en Pascal es:

```
PROCEDURE SeleccionDirecta;

VAR
  i,j,k:tIndice;
  Elemento:tElemento;
BEGIN
  FOR i:=1 TO N-1 DO
  BEGIN
    k:=i;
    Elemento:=Vector[i];
    FOR j:=i+1 TO N DO
      IF Vector[j]<Elemento THEN
      BEGIN
        k:=j;
        Elemento:=Vector[j];
      END;
    Vector[k]:=Vector[i];
    Vector[i]:=Elemento
  END

END;
```

El número de comparaciones entre claves va a ser independiente de la ordenación inicial de éstas. Respecto a si es natural o antinatural decimos que es indiferente, no es ninguna de las dos cosas. Podemos decir que se comporta de manera menos natural que el método de inserción directa.

$$C = 1/2\ (n^2 - n)$$

Respecto al número de movimientos decimos por norma general que sí es natural, aunque depende de la implementación.

El número de movimientos queda como sigue:

$$M_{min} = 3(n-1)$$
$$M_{max} = trunc(n^2/4) + 3(n-1)$$
$$M_{med} = n(\ln n + 0.577216)$$

No siempre es estable.

Los métodos más avanzados se muestran poco eficaces con escasos datos a ordenar

Como conclusión, el algoritmo de selección directa es mejor que el de inserción directa, aunque en los casos en los que las claves están inicialmente ordenadas el de inserción directa es un poco más rápido.

BURBUJA (INTERCAMBIO):

Se basa en el principio de comparar e intercambiar pares de elementos adyacentes hasta que quede ordenado. Se hacen repetidas pasadas sobre el vector, moviendo en cada una el elemento de clave mínima hasta el extremo izquierdo del mismo.

Si se mira el vector verticalmente, los elementos se consideran burbujas en un depósito de agua con pesos acordes con sus claves; de cada pasada sobre el array resulta la ascensión de una burbuja hasta el nivel de peso que le corresponde. De ahí le viene el nombre de método de la burbuja.

En el ejemplo la secuencia de ordenación queda de esta forma.

	i = 2	i = 3	i = 4	i = 5	i = 6	i = 7	i = 8
36	**02**	02	02	02	02	02	02
58	36	**12**	12	12	12	12	12
12	58	36	**17**	17	17	17	17
45	**12**	58	36	**45**	45	45	45
98	45	**17**	58	**36** →	**36**	36	36
17	98	45	**45**	58	**58** →	**58**	58
02	**17**	98	**70**	70	70	**70** →	**70**
70	70	**70**	98	98	98	98	98

El algoritmo en Pascal es el siguiente:

```
PROCEDURE Burbuja;
VAR
  i,j:tIndice;
  Elemento:tElemento;
BEGIN
  FOR i:=2 TO N DO
    FOR j:=N DOWNTO i DO
    BEGIN
      IF Vector[j-1]>Vector[j] THEN
      BEGIN
        Elemento:=Vector[j-1];
        Vector[j-1]:=Vector[j];
        Vector[j]:=Elemento
      END
    END
END;
```

Este método es estable. Si las dos claves son iguales no se intercambian.

El método es natural en cuanto a los intercambios. En cuanto a las comparaciones, si lo implementamos como arriba no afecta, pero si incluimos esta mejora (mirar el número de intercambios. Si no se producen el vector está ordenado y nos ahorramos n-2 pasadas) es natural.

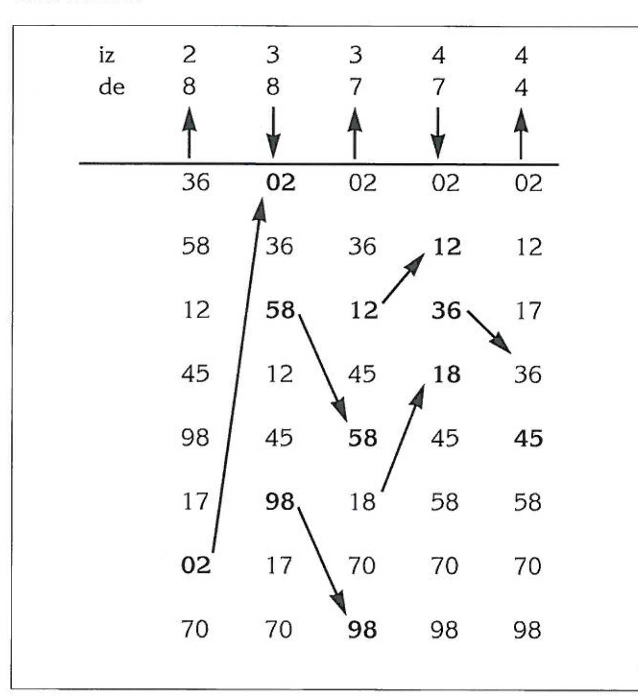
SACUDIDA (INTERCAMBIO):

Con el vector [2,3,4,6,1] y el método de la burbuja, de dos pasadas se ordena. Sin embargo, el vector [6,1,2,3,4,5] tiene que efectuar n-1 pasadas.

Para evitar cosas de este tipo lo que se hace es una pasada en cada sentido (bidireccional). Este es el método de la sacudida. Hace el mismo número de intercambios pero a veces hace menos comparaciones que el método de la burbuja.

Son de los peores métodos de comparación ya que hacen muchos intercambios y en distancias muy cortas.

En el ejemplo la secuencia de ordenación queda de esta forma.



El algoritmo en Pascal es el siguiente.

```
PROCEDURE Sacudida;
VAR
 j,k,iz,de:tIndice;
 Elemento:tElemento;
BEGIN
 iz:=2;
 de:=N;
 k:=n;
 REPEAT
  FOR j := de DOWNTO iz DO
   IF Vector[j-1]>Vector[j] THEN
   BEGIN
    Elemento:=Vector[j-1];
    Vector[j-1]:=Vector[j];
    Vector[j]:=Elemento;
    k:=j
   END;
  iz:=k+1;
  FOR j:= iz TO de DO
   IF Vector[j-1] > Vector[j] THEN
   BEGIN
    Elemento:=Vector[j-1];
    Vector[j-1]:=Vector[j];
    Vector[j]:=Elemento;
    k:=j
   END
  de:=k-1
 UNTIL iz > de
END;
```

El número de comparaciones en estos dos algoritmos es:

$$C = 1/2\,(n^2-n)$$

Y los movimientos:

$$M_{min} = 0$$
$$M_{med} = 3/4\,(n^2-n)$$
$$M_{max} = 3/2\,(n^2-n)$$

El algoritmo de selección directa es mejor que el de inserción directa

La ordenación por intercambio es inferior a la ordenación por selección o por inserción. El método de la sacudida se utiliza con ventaja cuando sabemos que el array está casi ordenado, lo que es raro en la práctica.

METODOS AVANZADOS

QUICKSORT (RAPIDO):

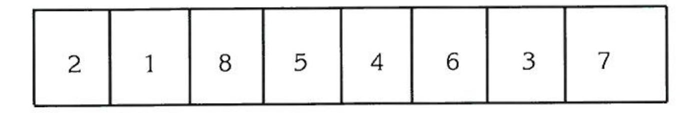
Está basado en el principio de intercambio. Es el método más eficaz que se conoce para ordenar vectores.

Se basa en el hecho de que los intercambios deben realizarse preferiblemente sobre distancias largas para que sean más efectivos. Para realizar la ordenación se toma de forma arbitraria un elemento cualquiera x; se examina a izquierda y a derecha hasta encontrar un ei <= x; se examina el vector de derecha a izquierda hasta encontrar un ej >= x y se intercambian ambos. Se continúa hasta que los dos procesos coincidan con el mismo elemento del vector. Se habrá conseguido dividir el vector en dos partes. Una a la izquierda de x con todos los elementos menores que x y otra a la derecha con todos los mayores o iguales. Repitiendo el proceso con cada uno de ellos hasta que tengan menos de dos elementos queda ordenado el vector.

La secuencia de ordenación del vector [2,7,8,5,4,6,3,1] sería:

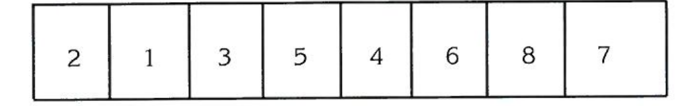
2	7	8	5	4	6	3	1

Se toma d como elemento de partida. Se intercambia el 7 con el 1.



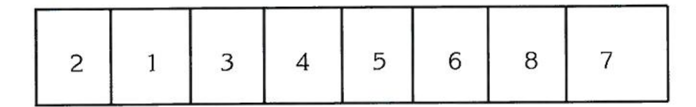
2	1	8	5	4	6	3	7

Se intercambia el 3 con el 8.



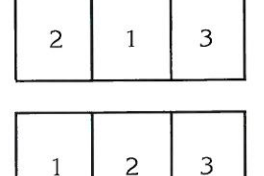
2	1	3	5	4	6	8	7

Se intercambian el 4 con el 5

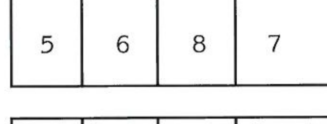


2	1	3	4	5	6	8	7

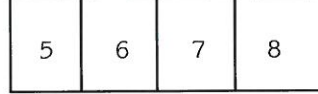
Aquí termina la primera parte. Ahora repetimos la misma operación con los dos subvectores obtenidos.



2	1	3
1	2	3



5	6	8	7



5	6	7	8

El algoritmo en Pascal correspondiente a este método es el siguiente.

```
PROCEDURE Rapido;
 PROCEDURE Sort(iz,de:tIndice);
 VAR
  i,j:tIndice;
  x,w:tElemento;
 BEGIN
  i:=iz;
  j:=de;
  x:=Vector[(iz+de) div 2];
  REPEAT
   WHILE Vector[i] < x DO
    i:=i+1;
   WHILE x < Vector[j] DO
    j:=j-1;
   IF i<=j THEN
   BEGIN
    w:=Vector[i];
    Vector[i]:=Vector[j];
    Vector[j]:=w;
    i:=i+1;
    j:=j-1
   END
  UNTIL i>j;
  IF iz < j THEN Sort(iz,j);
  IF i < de THEN Sort(i,de)
 END;
BEGIN
 Sort(1,N)
END;
```

El número de comparaciones que realiza este método es n. El número de intercambios esperado es aproximadamente n/6. En el caso más favorable -cuando elegimos la mediana como punto de partida- el número de pasadas que realiza para ordenarlo es de log n. Por tanto el número de comparaciones es n·log n y el de intercambios n/6·log n. Esto no puede esperarse que suceda siempre. Es más, la probabilidad de que esto suceda es de 1/n. Ahora sólo queda ver cómo se comporta en el caso más desfavorable. Este es su talón de Aquiles. En este caso se necesitan n procesos de subdivisión en vez de log n y el rendimiento es del orden de n2.

También tiene sus limitaciones cuando se trata de ordenar un vector con pocos elementos. Este método se muestra lento en estos casos.

CONCLUSIONES:

Hasta aquí se ha dado un buen repaso a los métodos de ordenación más conocidos. Como habrán podido observar las Matemáticas siguen siendo imprescindibles para el estudio de los algoritmos y estructuras de datos. No nos podemos alejar de ellas.

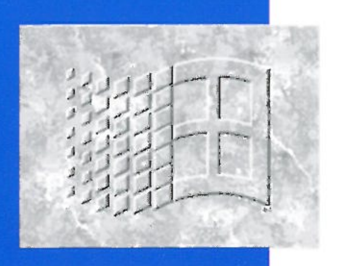
El QuickSort es el método más eficaz para ordenar vectores

También habrán podido ver que el método de ordenación más eficaz visto, el QuickSort o método de ordenación rápido, utiliza una técnica ya estudiada en el primer artículo de esta sección: la recursividad. Aquí hay un claro ejemplo de cómo para encontrar una solución óptima a un problema se ha recurrido a una técnica de programación no muy utilizada y poco conocida.

Habiendo estudiado en profundidad las características de todos los métodos de ordenación queda en sus manos elegir el más apropiado para resolver el problema que se le plantee. ■

TRATAMIENTO DE MENSAJES

Bernardo García



El principal problema que surge a la hora de aprender a programar en un entorno gráfico o *GUI* es el cambio en la forma de trabajo del mismo.
Mientras un programa clásico llama al sisitema operativo, en Windows el sistema es quien realiza esta tarea, enviando mensajes a las diversas aplicaciones.

En el anterior artículo se vio el aspecto que tiene un programa desarrollado para el entorno Windows, pero ¿ qué lo hace tan diferente ?

En la programación clásica los programas siguen una secuencia determinada de instrucciones. Solicitan datos al usuario y posteriormente realizan llamadas a las funciones del sistema operativo para procesar esos datos. Esta es una de las características que diferencia la programación en lenguaje C bajo Windows de la habitual en otros sistemas operativos. Aunque el lenguaje conserva su estructura, la mayoría de las funciones de la librería estándar no pueden ser utilizadas, siendo sustituidas por sus equivalentes en la librería de Windows. Un programa Windows también puede realizar llamadas, pero principalmente recibe mensajes comunicándole lo que sucede en el entorno.

Windows genera mensajes por cada evento que ocurre. Cuando el usuario mueve el ratón o pulsa una tecla, se genera uno. Esto también sucede con otro tipo de eventos, caso de un Timer, no necesariamente producidos por el usuario. Es muy importante comprender cómo se tratan internamente estas comunicaciones para poder programar en Windows.

EL BUCLE DE MENSAJES

Como se puede ver en el programa que acompaña a este artículo, un programa básico en Windows se compone de una función principal, llamada obligatoriamente *WinMain,* y otra de tratamiento de mensajes que suele recibir el nombre de *MainWndProc,* aunque en este caso el nombre es de libre elección.

Entre otras tareas, como son registrar la clase y crear la ventana, *WinMain* contiene el bucle de mensajes (Ver Listado 1). Esta porción de código es prácticamente idéntica en todos los programas Windows. A pesar de su brevedad cumple una misión fundamental para el funcionamiento de todo el sistema (ver figura 1).

Como se ha dicho anteriormente, cualquier evento o suceso que ocurre genera un mensaje en Windows. Estos son introducidos en una cola del sistema. Cada aplicación cuenta además con su propia cola. Aquellos que van dirigidos a una determinada ventana, pasan a la cola correspondiente del programa propietario de la misma.

La tarea del bucle de mensajes consiste en obtener los que afectan a la aplicación y enviarlos a la función de tratamiento, *MainWndProc,* que corresponda a esa ventana.

La función *GetMessage* es la encargada de recuperar los mensajes destinados a la aplicación. En primer lugar se comprueba la cola de la aplicación para buscarlos. Si esta cola estuviera vacía se continuará buscando en la cola de mensajes del sistema. Windows utiliza un sistema de multitarea 'cooperativo'. Mientras una aplicación está procesando uno, Windows la trata como si fuera la única tarea que existe. Solamente durante la llamada a *GetMessage* es donde se le quita el control para cederlo a otras aplicaciones que estén funcionando en ese momento. Si no hubiera ningún mensaje destinado a la aplicación en ninguna de las colas, ésta pasaría a estar inactiva hasta que se produzca uno para ella. Hasta ese momento Windows le cedería el control a otras aplicaciones o se quedaría a la espera de nuevos eventos.

Si se recibe un mensaje, *GetMessage* recupera su contenido y continua con él. Es entonces cuando actúa la función *TranslateMessage,* convirtiendo las pulsaciones de teclas en los caracteres que les corresponden, debido principalmente a que se generan dos mensajes, uno al pulsar la tecla y otro al liberarla. *TranslateMessage,* que sólo opera sobre los de este tipo, se encarga de convertirlos en uno solo de tipo WM_CHAR transformando el número de la tecla en el carácter ASCII que le corresponde según el estado de las teclas de Shift, Alt o Ctrl y agrupando los dos mensajes de pulsación y liberación de tecla.

Una vez recuperado, y en su caso convertido, *DispatchMessage* se ocupa de que el mensaje llegue a la función de tratamiento que le corresponda según la ventana. *MainWndProc* será quien se encargue de tratar ese.

Las colas de mensajes son de tipo FIFO (First In - First Out), con alguna pequeña variación. Es decir, los mensajes son tratados en el mismo orden en que llegan a la cola, el primero en llegar es el primero en salir. Existen excepciones a esta norma, como es el caso de WM_PAINT, que se genera cada vez que la ventana necesita ser repintada. Al ser una tarea secundaria, este mensaje va siendo dejado hasta que no queda ningún otro pendiente.

LISTADO 1: BUCLE DE TRATAMIENTO DE MENSJES

```
MSG msg
...
...
while (GetMessage(&msg, NULL, 0, 0))
  {
   TranslateMessage(&msg);
   DispatchMessage(&msg);
  }
...
...
```

Este proceso de recuperación y envío se repite hasta que *GetMessage* recibe un mensaje WM_QUIT que le indica que debe terminar la aplicación.

EL CORAZON DEL PROGRAMA

Hasta aquí aún no se ha realizado ninguna acción del programa. Todo estaba orientado a la comunicación de la aplicación con Windows. *WinMain* ya ha creado la ventana y está enviando los que se producen a *MainWndProc,* la función de tratamiento de mensajes y el verdadero corazón del programa.

Cada clase de ventana debe tener un función de este tipo, que se indica en la llamada a *RegisterClass* (ver mensajes.c en el disco) para tratar los mensajes enviados a la ventana.

En el listado 2 se puede observar la estructura de la función *MainWndProc.* Consiste en una estructura selectiva que realiza unas operaciones determinadas según lo que recibe. Esta función nunca es llamada directamente por el programa. Es *DispatchMessage* u otras funciones relacionadas con los mensajes quienes se encargan de llamar la.

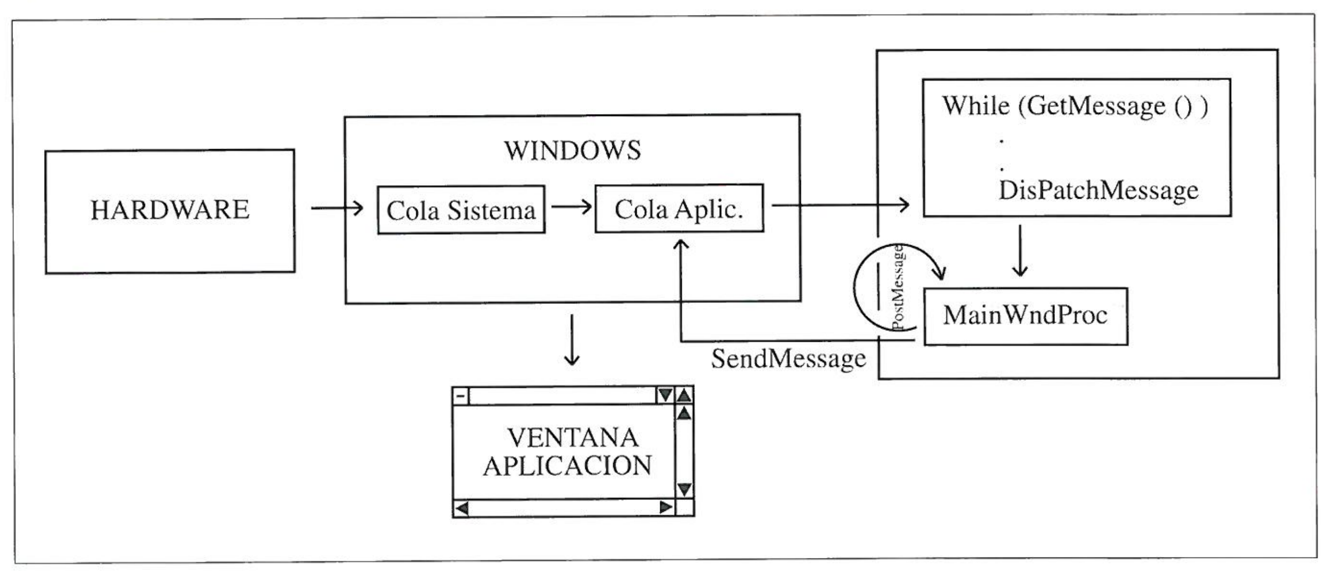


Figura 1.

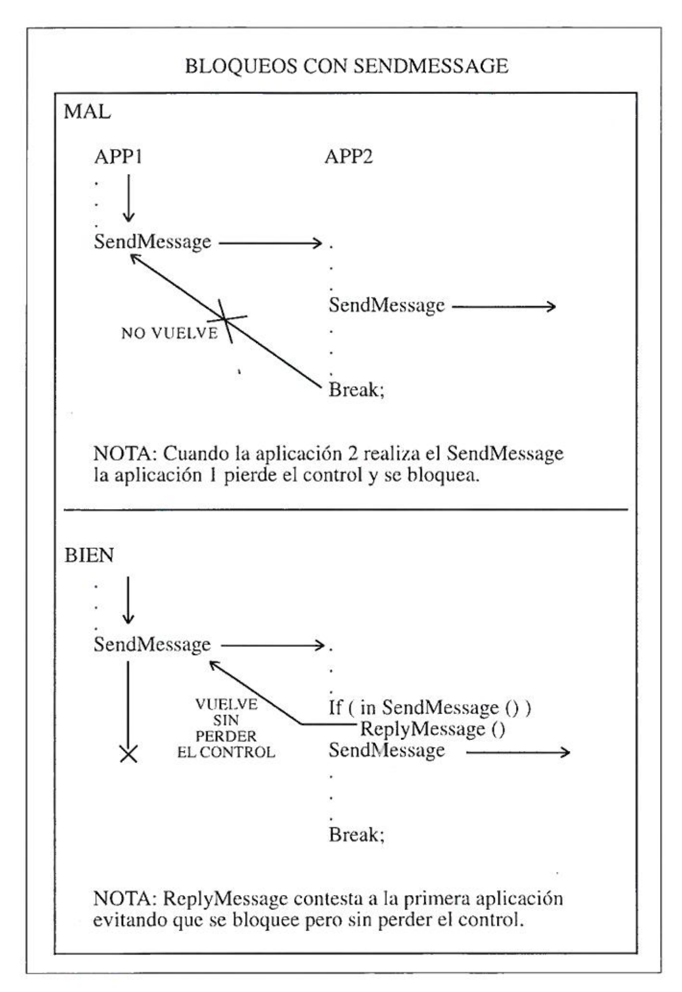


Figura 2

Si el recibido no es uno de los que se deben tratar, entonces debe ser devuelto al Windows para que lo haga, a través de la función de tratamiento por defecto, *DefWindowProc*. Esta operación garantiza que el mensaje siempre va a ser tratado por alguien.

En caso de recibir uno de los propios de la aplicación, el tratamiento del mismo comenzará en *MainWndProc*. Durante esta operación se elimina cualquier posibilidad de que Windows ceda el control a otra aplicación. Por esta razón los programas "educados" deben tratar de seguir unas normas para evitar así el bloqueo del sistema. Durante el tiempo que se esté tratando el mensaje, Windows no puede quitarle el control a la aplicación, por lo que los demás programas quedarían detenidos.

En primer lugar el tratamiento de cada mensaje debe ser lo más breve posible para volver cuanto antes al bucle de mensajes. Como esto no siempre es posible, si el proceso va a durar mucho tiempo la aplicación debe tenerlo en cuenta para ceder periódicamente el control de forma voluntaria.

La función *PeekMessage* sirve para averiguar si hay mensajes pendientes en la cola de la aplicación. Es similar en su funcionamiento a *GetMessage,* ya que durante esta función Windows puede ceder el control a otras aplicaciones, con la diferencia de que no extrae los mensajes de la cola, sólo los recupera para poder investigarlos. Si el que se obtiene tuviera que ser eliminado de la cola, el programa debería realizar el mismo proceso que en el bucle de mensajes, llamar a *GetMessage.*

Como se puede ver el tratamiento de los mensajes empieza a complicarse; no sólo se pueden leer sino que también se pueden "cotillear". Y aún más, hasta ahora se había visto que únicamente *DispatchMessage* enviaba mensajes a la función de tratamiento. Esta también puede enviarse a sí misma, y además puede hacerlo por dos métodos diferentes.

Lo lógico, visto el funcionamiento, sería pensar que todos los mensajes van destinados a la cola de la aplicación que les corresponde. Este método es el empleado por la función *PostMessage,* que envía un mensaje a una ventana pasando por la cola de la misma. Sin embargo existe otro método, el empleado por *SendMessage,* que consiste en llamar directamente a la función de tratamiento de mensajes de la ventana, igual que hace la función *DispatchMessage.*

Aunque el resultado de ambos métodos es el mismo, varía el camino seguido para conseguirlo. *PostMessage* emplea el mismo método que sigue el Windows, enviarlo a la cola de mensajes de la aplicación, para que sea el bucle de mensajes el que se encargue de recuperar y tratarlo. Este es el método más sencillo ya que parte del proceso, como *TranslateMessage,* ya está implementado en el bucle. *SendMessage,* sin embargo, envía el mensaje directamente a la función de tratamiento, saltándose todos los que están esperando en la cola. De esta forma el mensaje es tratado de forma inmediata, aunque en algunos casos hay que realizar tareas previas con él, antes de enviarlo.

El principal problema con *SendMessage* proviene de la pérdida de control de la aplicación y los bloqueos que se pueden producir al realizar llamadas consecutivas a esta función. Si una aplicación utiliza esta función cede el control al *MainWndProc* que va a tratar el mensaje, y no puede seguir ejecutándose hasta que esta última termine. Si la función que ha recibido la llamada, realiza a su vez otra operación del mismo tipo, la aplicación que realizó la llamada original nunca recuperará el control, quedándose bloqueada a la espera de que su mensaje sea tratado (Ver figura 2).

Para evitar que esto ocurra, existe una función que permite conocer cuándo la llamada es desde *DispatchMessage* o *SendMessage*. Se trata de *InSendMessage.* Si devuelve un valor que no sea cero, indica que se esta respondiendo a una llamada de tipo *SendMessage* y para evitar bloqueos se debería utilizar la función *ReplyMessage,* que responde a la aplicación original, pero sin devolverle el control, de forma que no queda bloqueada pero debe esperar su turno para volver a recibir el control.

En el listado 4 se pueden ver las funciones de Windows que se deben evitar utilizar cuando se está respondiendo a una llamada realizada con



LISTADO 2: ESQUELETO DE MAINWNDPROC

```
long CALLBACK MainWndProc(HWND hWnd,
UINT message, WPARAM wParam, LPARAM lParam)
{
 switch (message)
  {
   case WM_COMMAND:
                    ......
       break;
   case WM_PAINT:
                    ......
       break;
   case WM_DESTROY:    /* Mensaje para Destruir la
                       Ventana */
       PostQuitMessage(0);
       break;
       default:        /* Los mensajes no tratados
                       se los dejamos al Windows */
       return (DefWindowProc(hWnd, message, wParam,
             lParam));
  }
 return (NULL);
}
```

SendMessage. Es una buena práctica de programación comprobar, con *InSendMessage* y *ReplyMessage,* si se está en esa situación antes de utilizar estas funciones para evitar así futuros problemas a cualquiera que realice llamadas a la aplicación.

FLUJO DE MENSAJES

Después de haber visto el funcionamiento básico del mecanismo de los mensajes en Windows, se debe conocer cuál es el flujo habitual en cualquier aplicación.

En el listado 3 aparece la estructura de uno, que está compuesto por los siguientes campos:

* hWnd: es el identificador de la ventana a la que va dirigido.
* message: es el código.
* wParam y lParam: son los parámetros.
* time: indica el momento en que se produjo.
* pt: indica la posición del ratón cuando se produjo.

Windows posee varios cientos de mensajes predefinidos identificados por empezar con WM_, además de dar la posibilidad al programador de crear otros nuevos.

El primero que recibe una aplicación es el de WM_CREATE o WM_NCCREATE, enviado por la función *CreateWindow.* En este momento aun no existe la cola de mensajes, por lo que este es enviado directamente a la función *MainWndProc.*

Una vez creada la ventana de la aplicación, esta puede no ser visible, dependiendo del estilo de la misma. Para mostrar la ventana se emplea la función *ShowWindow,* que envía el mensaje WM_SHOWWINDOW. A su vez la misma función, enviándolo otra vez realiza la operación contraria, ocultar la ventana. Este, dependiendo de los valores de los parámetros realizará una u otra acción (ver listado 6) según lo que esté definido en windows.h.

Al igual que el anterior, *GetFocus* y *SetFocus,* a través de WM_ACTIVATE, activaran o desactivaran la ventana dependiendo del valor de sus parámetros.

Durante el proceso normal de una aplicación se irán sucediendo multitud de mensajes. Este flujo terminará cuando la aplicación reciba WM_DESTROY. Se produce cuando la aplicación realiza una llamada a *DestroyWindow* o cuando *DefWindowProc,* la función de tratamiento por defecto de Windows, recibe WM_CLOSE. El objetivo de WM_DESTROY es indicar a la aplicación que debe realizar las operaciones que considere oportunas para terminar. Una vez que las haya realizado será la propia aplicación la que termine, llamando a la función *PostQuitMessage* o enviándose WM_QUIT a sí misma. Esto permite que una aplicación se niegue a terminar, en caso de que esto ocurriera no hay ninguna forma de obligarla a terminar más que apagar el ordenador, ya que Windows a su vez no puede terminar mientras alguna aplicación continúe activa.

WM_QUIT es un caso especial. Este, nunca llegará a la función de tratamiento de mensajes, ya que *GetMessage* lo interpretará como indicativo de que ya no queda ninguno por procesar y terminará el bucle de mensajes. Por esta razón, antes de enviar WM_QUIT la aplicación debe cerciorarse de haber realizado todas las tareas necesarias antes de acabar.

De entre todos los mensajes que existen hay algunos que aparecen más habitualmente que otros.

El mensaje WM_CHAR se produce cuando es pulsada alguna tecla. Su función es transmitir a la aplicación el carácter de la tecla pulsada. Este, junto con WM_SYSCHAR, no son generados directamente por Windows, sino que son creados por la función *TranslateMessage* a partir de la información de estado del teclado.

Otro mensaje habitual es WM_PAINT. Windows no almacena información referente al contenido de cada

LISTADO 3: ESTRUCTURA DE UN MENSAJE TYPEDEF STRUCT TAGMSG

```
{
 HWND hWnd;                    /* Handle de la Ventana        */
 UINT    message;              /* Codigo de Mensaje           */
 WPARAM          wParam;  /* Valores especificos para el
                              mensaje */
 LPARAM          lParam;
 DWORD           time;    /* Hora en que se produjo el men-
saje  */
 POINT pt;                     /* Punto donde estaba el raton
                              situado */
} MSG;
```

LISTADO 4: FUNCIONES QUE PUEDEN CAUSAR UN BLOQUEO CON SENDMESSAGE

DialogBox
DialogBoxIndirect
DialogBoxIndirectParam
DialogBoxParam
GetMessage
MessageBox
PeekMessage
Yield

LISTADO 5: FUNCIONES RELACIONADAS CON EL TRATAMIENTO DE MENSAJES

Funciona	Descripción
CallWindowProc	Pasa la información de un mensaje a una funciona de tratamiento de una ventana
DispatchMessage	Pasa un mensaje a la funciona de tratamiento de una ventana especifica
GetMessage	Recupera un mensaje de la cola de la aplicación
GetMessageExtraInfo	Obtiene información adicional acerca de un mensaje de Hardware
GetMessagePos	Obtiene la posición del ratón cuando se produjo el mensaje
GetMessageTime	Obtiene la hora en que se produjo el mensaje
GetQueueStatus	Devuelve un valor identificando el tipo de mensajes, si los hay, en la cola de la aplicación
hardware event	Envía un evento de tipo Hardware a la cola de mensajes del sistema
InSendMessage	Averigua si la funciona de tratamiento de mensajes está procesando un mensaje enviado por otra aplicación a través de SendMessage
PeekMessage	Comprueba si existen mensajes en la cola de la aplicación.
PostAppMessage	Envía un mensaje a la cola de mensajes de una aplicación
PostMessage	Envía un mensaje a la cola de mensajes de una ventana
PostQuitMessage	Envía el mensaje WM_QUIT a la aplicación
ReplyMessage	Contesta a un mensaje enviado por otra aplicación sin devolverle el control
SendMessage	Envía un mensaje directamente a la funciona de tratamiento de una ventana o ventanas
SetMessageQueue	Crea una nueva cola de mensajes o cambia su tamaño
TranslateMessage	Convierte las teclas virtuales en caracteres del código ASCII
WaitMessage	Cede el control a otras aplicaciones

ventana, ya que esto consumiría gran cantidad de recursos. Por ello, cada ventana ha de saber cómo repintarse a sí misma. Cuando alguna zona de la pantalla necesita repintarse, Windows envía un mensaje a las ventanas que tienen alguna parte en el área no válida. Es la propia ventana la que debe repintarse en respuesta a éste. La aplicación también puede generar este mensaje cuando considere que hay que actualizar la información de la ventana, para lo que puede emplear la función *UpdateWindow*.

El mensaje tratado con mayor frecuencia es WM_COMMAND. Su función es agrupar las ordenes enviadas a la ventana, y que se refieren específicamente a la aplicación. Mientras los demás que se han visto responden a las órdenes o a tareas especificas de Windows, como puede ser repintar la ventana o mostrarla, este mensaje se encarga de realizar las acciones propias de la aplicación. Mientras el resto de la aplicación es muy similar a cualquier otro programa para Windows, el tratamiento de WM_COMMAND es el que sufrirá más cambios entre una y otra aplicación.

Cabe mencionar, como último, WM_USER. Este constituye la frontera entre los mensajes propios de Windows, cuyo código no puede ser alterado, y los definidos por el usuario. Gracias a la posibilidad de enviar mensajes, una aplicación puede definir, enviar y recibir los suyos propios. El único requisito para esto, es utilizar como mensaje más pequeño WM_USER, siendo el código WM_USER+0.

En WM_COMMAND y WM_USER se agrupa el código relacionado específicamente con la aplicación.

Para ilustrar un poco más en profundidad el tema de los mensajes, se ha ampliado la aplicación del mes anterior. Esto permite mostrar la utilidad de la aplicación genérica. Basándose en ella se han incorporado unas pocas líneas de código, permaneciendo el resto inalterado. Aparte de unos cambios en los textos y mensajes informativos de la misma, la aplicación sólo ha crecido en las líneas que se muestran en el listado 7 para tratar tres nuevos.

Por orden de complejidad se puede observar que WM_SIZE únicamente realiza una llamada a la función *PostMessage* destinada a enviar el mensaje WM_PAINT a la aplicación. WM_SIZE se produce cada vez que la ventana cambia de tamaño, al maximizar o minimizar, la pulsación del botón correspondiente genera uno de este tipo.

WM_MOVE realiza una misión más complicada. Cuando la ventana es movida se genera este mensaje para indicar la posición final de la ventana. La operación que la aplicación va a realizar es la misma que en WM_SIZE, con la diferencia de la función empleada para enviarlo, que en este caso es *SendMessage*. Para evitar que la aplicación pueda bloquearse al llamar repetidamente a *SendMessage*, se realiza una comprobación con las funciones *InSendMessage* y *ReplyMessage*. Tras esta operación se envía el mismo mensaje que se empleo antes, WM_PAINT.

LISTADO 6: POSIBLES ACCIONES DE SHOWWINDOWS

Valor	Acción
SW_HIDE	Oculta la ventana
SW_MINIMIZE	Minimiza la ventana
SW_RESTORE	Activa y muestra la ventana en su estado original
SW_SHOW	Activa y muestra la ventana en su estado actual
SW_SHOWMAXIMIZED	Activa y muestra la ventana maximizada
SW_SHOWMINIMIZED	Activa y muestra la ventana minimizada en forma de icono
SW_SHOWMINNOACTIVE	Muestra la ventana minimizada en forma de icono pero sin activarla
SW_SHOWNA	Muestra la ventana en su estado actual pero sin activarla
SW_SHOWNOACTIVATE	Muestra la ventana en su estado más reciente pero sin activarla
SW_SHOWNORMAL	Activa y muestra la ventana en su estado actual

LISTADO 7: MODIFICACIONES DEL CODIGO RESPECTO DEL MES ANTERIOR

```
long CALLBACK MainWndProc(HWND hWnd,
UINT message, WPARAM wParam, LPARAM lParam)
{
 HDC        hDc;
 PAINTSTRUCT ps;
 RECT       rect;
 ...
 ...
   case WM_PAINT:
      hDc = BeginPaint(hWnd, &ps);
      GetClientRect(hWnd, &rect);
      DrawText(hDc, "Mensaje de WM_PAINT", -1, &rect,
              DT_SINGLELINE | DT_CENTER | DT_VCENTER);
      EndPaint(hWnd, &ps);
        break;
   case WM_MOVE:
      if (InSendMessage())
         ReplyMessage(NULL);
      SendMessage(hWnd, WM_PAINT, 0, 0L);
        break;
   case WM_SIZE:
        PostMessage(hWnd, WM_PAINT, 0, 0L);
        break;
 ...
 ...
}
```

El último de los mensajes muestra cómo la programación en C varía de forma significativa en este entorno. El tratamiento que se pretende realizar es el de escribir un texto centrado en el área de la ventana. Esta operación no es tan simple como pudiera parecer, pues la ventana puede cambiar de tamaño o moverse. En primer lugar, las funciones *BeginPaint* y *GetClientRect* determinan el área de la pantalla ocupada por la ventana, en relación a la cual se deberá centrar el texto. *DrawText* se encarga de realizar esta operación. Se puede observar que no se realiza ninguna operación para determinar la posición del texto, únicamente se le indica a través de DT_CENTER y DT_VCENTER la intención de que el texto aparezca centrado vertical y horizontalmente en la ventana. Para terminar la operación se llama a la función *EndPaint* que da por concluidas las operaciones y actualiza el área de pantalla con su nuevo contenido. La aparente complejidad de estas operaciones no es nada si se compara con las que se deberían realizar si hubiera que controlar todos los factores, como el tamaño, posición y estado de la ventana, la longitud del texto, división de palabras, etc.

Una vez se ha compilado el programa, la ejecución del mismo nos permitirá ver cómo funcionan las operaciones que se han añadido. Si se cambia el tamaño de la ventana utilizando los bordes de la misma, el texto permanece en su sitio, ya que el mensaje WM_SIZE se produce con los botones de maximizar o minimizar. Si se pulsa el botón de maximizar, el texto permanece centrado en la pantalla.

Para observar mejor el funcionamiento, se puede cambiar el tamaño de la ventana, usando los bordes, de forma que el mensaje no se centra automáticamente. Con el texto descolocado, al mover la ventana se genera WM_MOVE, que envía WM_PAINT, forzando a que se realice la operación de centrado.

CONCLUSION

Se ha visto cómo se relaciona internamente Windows con las aplicaciones que están funcionando en este entorno, y cómo utilizando de base una aplicación genérica se puede ir complementando para añadirle las instrucciones necesarias sin tener que reescribir toda la aplicación.

Una vez resueltos estos dos puntos, la mayor dificultad de la programación en Windows la constituye la gran cantidad de mensajes y funciones a disposición del programador. Realizar una aplicación, ahora que se ha visto el corazón de la misma, es una simple tarea de colocar piezas en su sitio. Sólo hace falta encontrar y saber cómo manipularlas. ■

EL PROBLEMA DE LA REDUCCION DE COLORES

Juan Ramón Lehmann

Las imágenes con una profundidad de colores de 24 bits dan una apariencia de gran realidad sobre nuestro monitor, pero el almacenamiento y tratamiento de éstas es caro en memoria y tiempo. Una imagen almacenada a 320*200 pixels de True Color, ocuparía 320*200*3, lo que nos da 192.000 bytes, mientras que a 256 colores, ocuparía solamente 64.000 bytes. Esto es causa suficiente para persuadirnos de reducir los colores en el tratamiento de imágenes.

CONCEPTOS BASICOS

Los colores en nuestro monitor son generados a partir de cantidades de tres básicos: Rojo, Verde y Azul (RGB-> Red, Green, Blue). Estas se mezclan para formar un color determinado que será visualizado en el monitor. Para gestionar estas mezclas se construye una paleta que no es más que una tabla indexada, en la cual cada índice apunta a un color (mezcla de RGB) determinado.

La longitud de esta paleta, viene determinada por la profundidad del color, ya que según los bits empleados en representar estas cantidades de mezclas será más amplia la paleta o no. Si empleamos una profundidad de 8 bits por componente, tendremos una imagen con 16 millones de colores, ya que:

8 bits (Red) + 8 bits (Green) + 8 bits (Blue)= 24 Bits, lo cual nos da una representación de 224 colores o lo que es lo mismo a 16,77 millones de colores posibles.
Si empleásemos una profundidad menor: 5 bits (Red) + 5 bits (Green) + 5 bits (Blue)= 15 Bits, que nos proporcionaría una representación de 215 ó 32.768 colores, etc...

Podemos imaginarnos el color RGB como un cubo tridimensional (Figura 1), en cuyos vértices descansan los colores puros, y todo el volumen es la representación gráfica sobre un eje de coordenadas en el espacio de todos los colores posibles. El componente Red será el eje de la X, Green el de la Y y Blue el de la Z, de tal manera que cada color tiene una posición determinada por sus componentes dentro del cubo.

El punto {12,12,0} (R=12,G=12,B=0) representará un color que se ubicará dentro del mencionado cubo. Este color, como puede imaginarse, estará dentro del espacio del cubo, muy cerca del color {13,12,0} (basta trasladar los componentes como si de coordenadas espaciales se tratase); lo que lógicamente nos hace suponer

Existen muchos algoritmos que han sido desarrollados para solventar el problema de la representación de imágenes True Color (24 bits ó 16 millones de colores) en dispositivos con capacidad inferior, como el que nos ocupa en este artículo, el árbol octal octree.

que estos colores serán muy parecidos, y de hecho lo son. La clave del problema de la reducción de colores, se solventa mediante esta teoría:

Cuanto más cercanas estén las coordenadas en el espacio del cubo RGB, más parecidos serán los colores representados por dichas coordenadas.

La cercanía de los anteriores ejemplos de colores, era fácilmente deducible, ya que sólo habíamos incrementado el eje de las X en una unidad. Pero, ¿cómo podríamos calcular la distancia entre 2 puntos en casos menos obvios, como {12,3,6} y {0,255,6}?

La composición de colores RGB puede representarse como un cubo en 3D

En algunos algoritmos se calcula mediante la formula de distancia de Euclides, que precisamente sirve para determinar la distancia entre dos puntos en el espacio:

D(x,y,z)=raíz cuadrada((X1-X2)al cuadrado+(Y1-Y2)al cuadrado+(Z1-Z2)al cuadrado))

Naturalmente, se deben sustituir las coordenadas X,Y,Z por R,G,B. Esta sería una manera aceptable de calcular su distancia, aunque cara en tiempo de ejecución.

Otros algoritmos usan otras soluciones simplificadas (en el aspecto de que consumen menos tiempo de ejecución) como la distancia Mannhattan, definida como:

D(x,y,z)= raíz cuadrada((X1*X2)+(Y1*Y2)+(Z1*Z2))

Lo que nos da un margen de error mayor que la distancia Euclídea.

Nosotros utilizaremos un criterio de quantización según la distancia en el cubo basándonos en los bits de cada componente. La función Testbit realizará esta tarea como veremos más adelante.

CONCRETANDO GENERALIDADES

Para elegir una serie de colores, en este caso 256 de entre 16,77 millones, deberemos basarnos en el criterio de cercanía de esos colores. Dividiremos el cubo RGB en 256 cubos menores, los cuales intentarán representar la totalidad de colores existentes en la imagen. Cada subcubo agrupará mediante un color representativo todos los colores en él incluidos. Estos subcubos serán adaptables a los colores que la imagen a tratar posea, para lo cual deberemos hacer un histograma de la misma.

ALGORITMOS QUANTIZADORES

Tomando como ejemplo una foto original en True Color (Imagen 1a), vamos a revisar superficialmente algunos de las soluciones más conocidas.

Para realizar la tarea antes descrita, se han desarrollado varios algoritmos. Nos encontramos con algoritmos como el Popularity algorithm (Heckbert 1982), que selecciona los K colores más frecuentes del histograma de la imagen. La calidad de las imágenes resultantes de la quantización mediante este algoritmo depende en gran medida de la imagen. Puede verse esta calidad comparativamente en la Imagen 1b.

Otro como el Median Cut algorithm (Heckbert 1982), trata de seleccionar K colores, intentando que cada color represente aproximadamente el mismo número de pixels (Imagen 1c).

También está la quantización uniforme. Con este algoritmo, se intenta representar la imagen tratada mediante una paleta ya predefinida, seleccionando colores por su cercanía en el espacio a los colores predefinidos. (Imagen 1d).

Por último, está el Octree algorithm (Michael Gervautz y Werner Purgathofer), que trata de seleccionar K colores, de tal manera que estos representen a la totalidad de colores existentes en la imagen original (Imagen 1e).

Existen otros muchos algoritmos, como el de la minimización de la varianza (Wu), el de la aproximación Floyd y Steinberg, el Optimized Through Color-Map Manipulation de Dale Schumacher, etc.

Para una mejor visión en conjunto de los anteriores algoritmos, se incluye tabla comparativa de tiempo, espacio y calidad resultante de cada uno de ellos.

EL ARBOL OCTAL (OCTREE)

Definiremos una estructura de fácil y rápido acceso para esta tarea (Figura 2a). Esta estructura nos dejará determinar fácilmente la cercanía de los colores en el espacio, así como su representativa.

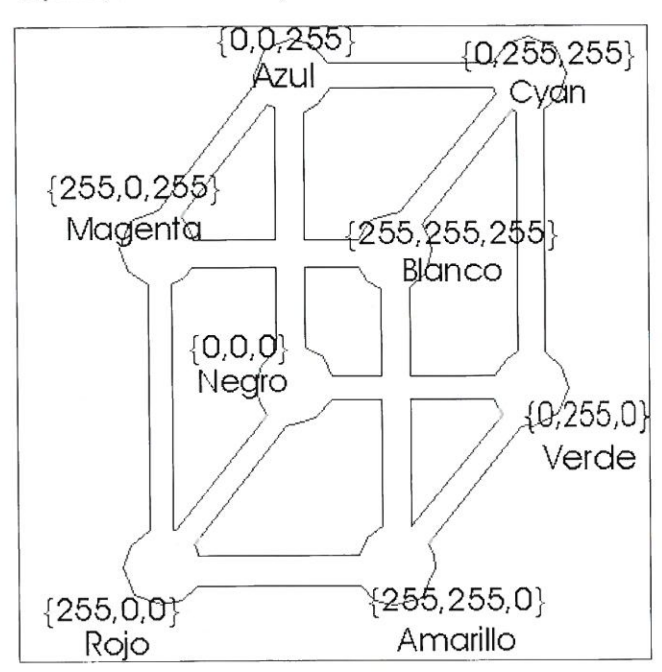


Figura 1: Cubo RGB.



Imagen 1a: Imagen original a 16,77 millones de colores en 24 bits.

El sistema de clasificación y evaluación de las representativas, será administrado como un Arbol, el cual tendrá unos puntos terminales (hojas), de profundidad 8. Estas tendrán una representación exacta del color procesado en ese instante. Los nodos intermedios (o ramas) representarán subcubos del cubo RGB principal. Dependiendo de la profundidad podremos saber cuál es exactamente el color representado para ese triplete de RGB, bastando simplemente mirar al puntero apuntado por él.

Como tarea primordial deberemos leer la imagen una primera vez, con objeto de evaluar las representativas. Una vez hecho esto, crearemos la tabla o paleta para la



Imagen 1b: Imagen quantizada mediante el algoritmo popular a 256 colores.

imagen a visualizar, y con posterioridad, paletizaremos la imagen en pantalla, con lo que finalizará la tarea del algoritmo.

Al comenzar la lectura de la imagen, introduciremos el color exacto en un nivel de profundidad 8, es decir, una hoja, actualizando los valores de RGB sumatorios, así como los campos anexos, con objeto de crear un histograma y obtener los datos necesarios que después utilizaremos para calcular el color representativo de ese subcubo. Cada color es introducido en el Arbol exactamente en su profundidad 8, pero en ningún momento dejaremos que haya más de K+1 colores. Caso que hubiera más de K+1 colores representados, actualizaremos el Arbol sustituyendo la hoja por su representativo, así hasta terminar la lectura secuencial de TODOS los tripletes (RGB) de la imagen a visualizar. En la Figura 2b, podemos ver el ejemplo de inclusión de algunos tripletes de muestra en dicho Arbol.

Nótese que la inclusión en el Arbol se realiza mediante la función Testbit, que lo único que hace es comprobar bit a bit cada color y formar un índice que se refleja en la asignación del puntero correspondiente en el array *next* de la estructura octree actual. Si se trasladan los valores decimales a binarios en la figura 2b, se podrá ver claramente lo mencionado.



Imagen 1c: Imagen quantizada mediante el algoritmo "Median Cut" a 256 colores.

Dado que en ningún momento hay más de K+1 colores en el Arbol, lo que estamos consiguiendo es reducir el número de colores originales al número especificado (K), ya que al alcanzar el límite definido en la constante, el algoritmo reduce automáticamente sacando el nodo a reducir del array reducelist.

Una vez leída toda la imagen, disponemos de una estructura que tiene K+1 colores representativos de los originales. Ahora deberemos recorrer el Arbol secuencialmente para calcular los representativos propiamente dichos.

Dado que hemos estamos actualizando los campos *rgbsum,* disponemos de la suma acumulada por com-

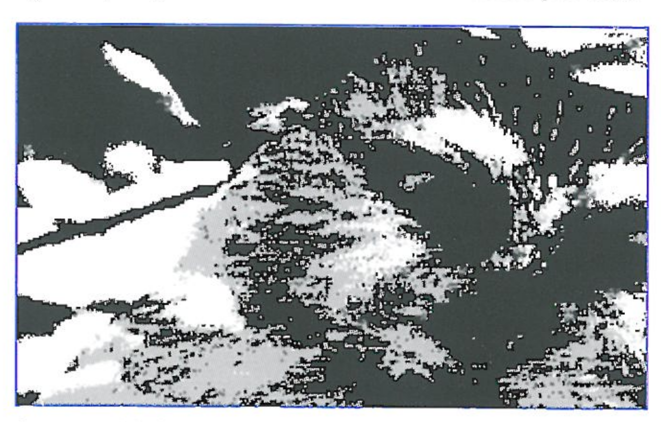


Imagen 1d: Imagen quantizada mediante el algoritmo Uniforme a 8 Colores (los vértices).





Imagen 1e: Imagen quantizada mediante el algoritmo "Octree" a 256 colores.

ponentes de cada triplete en su quantización por esa rama. También tenemos la cantidad de pixels por ese subcubo representados, luego entonces, para obtener el más representativo, nos basta hacer una media ponderada, es decir:

Representativo=RGBSum/NumerodePixels ->para cada componente sumatorio

Una vez hemos hecho esto en cada uno de los subcubos, debemos asignarle un índice para construir la paleta. Podemos hacerlo secuencialmente (si bien en

En el cubo RGB, al acercarse a los vértices, los colores son más parecidos

este caso lo hacemos paralelamente al calculo de su representativo), ya que no importa el índice que le asignemos, puesto que para paletizar la imagen (en una segunda lectura), usaremos el mismo criterio de localización en el Arbol que para su quantización.

Ya habiendo terminado el Arbol, deberemos construir la paleta. Nótese que en la función incluida para la tarea se introduce un factor de corrección, que nos quantizará con mayor o menor claridad la imagen. Este factor hace que el representativo se desplace más hacia el centro o no, lo que da una sensación de colores más quemados u oscuros respectivamente. Se aconseja utilizar un factor de corrección .2.

Una vez terminado el proceso anterior paletizaremos la imagen de acuerdo con el Arbol octal ya definido, y el resultado es el representado por las imágenes adjuntas.

ANEXOS

El algoritmo aportado, está implementado para compilar directamente desde Borland C++ 3.1, pero su conversión a otros compiladores no es demasiado complicada. Watcom y Microsoft disponen de funciones análogas a las de Borland, que bastaría sustituir. Quizá plantease un problema mayor pasarlo a otros compiladores tipo GNU o similar, pero aún en esos compiladores ANSI la compatibilidad con este código es muy elevada.

El listado también incluido viene comentado, de tal forma que sea de fácil comprensión al programador que desee implementarlo en su computadora.

Como puede observarse, basta con cambiar la constante MAXCOLORS para poder reducir de N colores a MAXCOLORS. El modo gráfico no es problema, ya que se puede cambiar también a discreción del programador.

FIGURA 2a

```
typedef struct node * OCTREE;
struct node {
        unsigned char leaf;
        unsigned char level;
        unsigned char colorindex;
        unsigned char children;
        unsigned long colorcount;
        struct colorsum rgbsum;
        OCTREE nextreduceable;
        OCTREE next[8];
        };

Donde:
leaf            => hoja, último nivel de cada branch del arbol.
level           => nivel actual.
colorindex      => indice de la paleta final.
children   => criterio para próximo a reducir.
colorcount   => contador de pixel/color por esa branch/hoja.
rgbsum          => campo para suma por componentes.
nextreduceable => puntero a reducir cuando >MAX_COLORS.
next[8]         => array de siguientes branchs/hojas.
```

FIGURA 2b

```
Testbit(a,i) —> ((a)>>i)&1) ; para testear bit a bit
   C1={2,0,0}
   C2={0,0,0}
   C3={200,1,64}
   N0B0H0C1     N0B0H0C2     N0B0H0C3
   N1B0H0C1     N1B0H0C2     N1B4H0C3
   N2B0H0C1     N2B0H0C2     N2B5H0C3
   N3B0H0C1     N3B0H0C2     N3B0H0C3
   N4B0H0C1     N4B0H0C2     N4B0H0C3
   N5B0H0C1     N5B0H0C2     N5B4H0C3
   N6B0H0C1     N6B0H0C2     N6B0H0C3
   N7B4H0C1     N7B0H0C2     N7B0H0C3
   N8B0H1C1     N8B0H1C2     N8B2H1C3 ->Exit
```

Para simplificar la representación gráfica de esta inclusión en el arbol octal, hemos codificado los graficos de la siguiente manera:

Nivel 1 Branch 5 Hoja 0 Color 1 —> N1B5H0C1

	Memoria	Búsqueda de representativas	Paletizado	Resultado
Quantización uniforme	0	O(K)	O(N)	Pobre
Arbol octal (octree)	O(K)	O(N)	O(N)	Buena
Algoritmo popular (Popularity Algorithm)	O(D)	O(N+ D.1d K)	>O(N)	Depende de la imagen
Median Cut	O(D)	O(N+ D.1d K)	O(N.1d K)	Buena

N= Numero de pixels en la imagen tratada.
K= Numero de colores representativos.
D= Numero de colores diferentes en la imagen tratada.

El programa se ha desarrollado con objeto de visualizar una imagen grabada en TGA tipo 2 en nuestro monitor, pero es fácilmente adaptable a conversión de formatos, reductor para tipos de TGA, etc.

CONCLUSION

Se añade como complemento a este artículo, un driver SVGA que permite al compilador de Borland representar en modo gráfico hasta 256 colores, por motivos obvios. Este driver es Shareware, con lo que esto implica. El programa de ejemplo incluido lee una imagen TGA tipo 2 y la muestra por pantalla utilizando este driver.

Espero que este articulo haya arrojado algo de luz sobre el tratamiento digital de la imagen, en particular sobre la reducción de colores. ■

BIBLIOGRAFIA

Die Elemente Der Mathematik (Reidt-Wolff)
The Digital DarkRoom
Graphics Gems I (Glassner-Academic Press)
Geometrie und eben Trigonometrie (Reidt-Wolff)
Graphics Gems II (Arvo-Academic Press)
Computer Graphics(Foley, van Dam-Addison Wesley)
y algunos artículos relacionados...
VGA Palette Mapping using BSP trees (Mark Betz-DDJ)
Color Models (Bruce Schneier-DDJ)
Median Cut Color Quantization (Anton Kruger-DDJ)
Digital Filtering Tutorial for Computer Graphics,Part II (Alvy Ray)

CORREO LECTORES

Para cualquier duda referente a los temas/árticulos tratatos en la revista, escriba una carta a:

Sólo Programadores
Referencia: *Correo Lectores*
TOWER COMMUNICATIONS
C/ Marqués de Portugalete, 10 Bajo
Madrid 28027

Sus cartas serán atendidas directamente por los autores de cada sección.

COMO SUSCRIBIRSE A



uscríbase enviando este cupón por correo o fax (91) 320.60.72, o llamando al teléfono (91) 741.26.62 Horario 9 a 14 y 15:30 a 18:30 h.

seo suscribirme a la revista SOLO PROGRAMADORES acogiéndome a la siguiente oferta:

1 año + 1 regalo (Filtro monitor) **por sólo** 10.995 ptas.

ombre y apellidos..Domicilio..Población..........................

ovincia...C.P ....Fecha de nacimiento.............................Profesión...

ORMA DE PAGO:

] Cheque a nombre de TOWER COMMUNICATIONS S.R.L., que adjunto.

] Contra-reembolso del importe más gastos de envío.

] Giro Postal (adjunto fotocopia del resguardo).

] Con cargo a mi tarjeta VISA nº ....

Fecha de caducidad de la tarjeta.................Nombre del titular, si es

into...

Firma:

] Domiciliación bancaria.

Señor Director del banco .... Agencia ....

Dirección del banco .... Población ....

Ruego a vd. que se sirva cargar en mi ☐ cuenta corriente ☐ libreta de ahorro número..

Rellena este cupón y envíalo a: TOWER COMMUNICATIONS SRL C/ Marques de Portugalete 10, Bajo 28027 Madrid.

LA POTENCIA DE LAS MACROS

Oscar García Reyes

En este artículo, mediante un ejemplo práctico, se tendrá posibilidad de ver cómo se pueden crear nuevas opciones en una aplicación Ms-Windows, en este caso Excel, a través de las diferentes herramientas de macros, quedando perfectamente integradas en su entorno.

COTILLEANDO A LOS VECINOS

Cuando se observa a la mayoría de las personas utilizar un ordenador, vemos que el noventa por ciento del tiempo lo dedican a realizar tareas repetitivas.

Sorprendentemente, el uso del ordenador es considerado por una gran parte de usuarios como uno de los trabajos más rutinarios a los que se enfrenta. Los motivos son, por una parte, que se utiliza un subconjunto muy pequeño de las posibilidades que las herramientas les brindan y por otra parte, que las instrucciones siguen encadenamientos que requieren la repetición de varias acciones para realizar el comando complejo que necesita una y otra vez. Este uso aberrante del ordenador es provocado porque se utilizan programas no especializados para el trabajo concreto que debe realizarse.

Es también asombrosamente alto el número de potenciales usuarios que se ven atemorizados por los ordenadores, debido a la terminología que en ellos se utiliza y de los procedimientos que deben seguir. "Arrancar un programa", "Guardar un fichero" o "Realizar un backup" pueden ser las palabras mágicas para que éstos consideren el ordenador como una gran fuente de problemas de la cual no quieren beber. Quieren encontrarse en un entorno y con una terminología que sientan familiares: "Cliente", "Acabar" y "Pesetas".

Y SE HIZO EL MACRO

Para resolver todos los problemas anteriores, surgieron los macros. Un macro es, básicamente, una colección de actuaciones que se realizan sobre un programa a una sola acción del usuario. Originariamente, se podía también resumir como una serie de teclas que son reproducidas por el ordenador al pulsar una combinación de las mismas especial.

Afortunadamente, la potencia de este tipo de actuaciones ya se vió mucho tiempo atrás, y prueba de ello es el servicio 5H de la interrupción 16H de la ROM BIOS (PC/XT a partir de 10/1/86, PC/AT desde el 15/11/85), o la función 0AH del DOS.

Una vez realizada la programación básica de una aplicación, se dan los retoques finales que normalmente afectan a la vistosidad y facilidad de uso. Pero, ¿y si se puede evitar la programación y pasar directamente a los retoques? Se consigue a simple vista un ahorro importantísimo de tiempo y recursos.

Con la llegada de Ms-Windows, aparece la grabadora capaz de conjugar funciones con teclado o ratón y con ella las tareas repetitivas se convierten en un "gracioso juego" en el que simplemente se observa como se van realizando tareas complejas. Sin embargo, el verdadero jugo se obtiene a partir de dar a los programas una forma sencilla de acercar al usuario toda la potencia de los macros. Cualquier aplicación potente de Ms-Windows ofrece posiblidades para usar esta herramienta.

Al observarse el rendimiento increíble que se podía lograr, se ha ido desarrollando más y más esta habilidad y de hecho, en las sucesivas versiones de Excel, la utilidad de macros ha ido variando hasta convertirse en la herramienta de programación que es hoy día. Secuencias de instrucciones de Excel, tareas repetitivas, la personalización del interface de usuario, e incluso el manejo y control de cualquier otra aplicación Ms-Windows a través del DDE y OLE dan idea del entorno en el que se mueve el programador.

Pero una imagen vale más que mil palabras; por ello se desarrollará completamente una aplicación sobre uno de los entornos de macros más potentes hoy día, como es el Excel 4.0. Hay que decir que la versión 5 de este programa añade un elemento importante a la programación, como es su posibilidad de integración con el Visual BASIC, si bien, todo lo que aquí se desarrolle es completamente compatible con dicha versión, y el uso de elementos del BASIC cae ya fuera de las pretensiones de este artículo.

GATEANDO AL PRINCIPIO

El programa a realizar cubre una necesidad muy común en las empresas. Se mantendrá una agenda de

TABLA 1: LISTA DE FUNCIONES UTILIZADAS

ABRIR?	Va al cuadro de dialogo para abrir.
AGREGAR.BARRA	Crea una nueva barra de men-.
AGREGAR.MENU	Crea un nuevo título en la barra de menús.
ALERTA	Informa o pide confirmación al usuario.
ANCHO.COLUMNA	Selecciona el ancho de la columna Excel.
ARCHIVO.CERRAR	Cierra un archivo.
CUADRO.DE.DIALOGO	Llama a un cuadro personalizado.
DEFINIR.NOMBRE	Define un nombre.
DETENER	Aborta la ejecución de macros.
DIRECTORIO	Selecciona el directorio activo.
EDICION.ELIMINAR	Elimina una fila, columna, celda,...
ELIMINAR.BARRA	Elimina una barra de menú.
ELIMINAR.NOMBRE	Elimina un nombre.
ESNUMERO	Comprueba si el argumento es un número.
ESTABLECER.VALOR	Coloca un valor en la hoja de macros.
ESTEXTO	Comprueba si el argumento es un texto.
FILA	Devuelve la fila de la referencia dada en el argumento.
FORMATO.FUENTES	Selecciona la fuente de letra y sus características.
FORMULA	Coloca un valor en la celda activa.
GUARDAR.COMO	Guarda una hoja.
INDICAR.DOCUMENTO	Devuelve información del documento (nombre, ...).
INTERRUMPIR	Interrumpe el bucle actual.
INTRODUCIR	Pide un valor al usuario (texto, número,...).
LARGO	Determina el largo de un texto.
LLAMAR.POR.TECLA	Prepara una interrupción por teclado.
MOSTRAR.BARRA	Muestra una barra como activa.
MOSTRAR.BARRA.HERRAMIENTAS	Muestra u oculta una barra de herramientas.
NO	No lógico.
NUEVO	Crea una hoja nueva.
REFTEXTO	Convierte una referencia en un texto.
SALIR	Salir de Excel.
SELECCIONAR	Selecciona la celda activa.
SUSTITUIR	Sustituye un texto con otro.
TEXTOREF	Convierte un texto en una referencia.
VALREF	Devuelve el valor de la celda referenciada en el argumento.
VENTANA.MAXIMIZAR	Maximiza la ventana activa.
VENTANA.MINIMIZAR	Minimiza la ventana activa.
VOLVER	Acaba el macro.
Y	Y lógico.

clientes en la cual se podrán añadir, eliminar, buscar o imprimir los datos de los clientes. Además, se verán diferentes posibilidades de ampliación de dicho programa básico. El estilo de programación será sobre todo didáctico, y por tanto, es posible que algunas funciones puedan ser optimizadas en revisiones posteriores por el propio lector, una vez vayan conociendo más los trucos y la potencia del propio lenguaje.

Con estas herramientas, parece lógico utilizar la técnica del prototipado, es decir, a partir del resultado final (visualmente hablando, se rellenan los puntos de programación que sean necesarios hasta conseguir el programa deseado, si bien aquí éste se irá abordando según la complejidad de sus diferentes partes.

Un macro es una serie de actuaciones repetitivas sobre un programa

"HELLO WORLD"

El Excel exige usar un tipo de hoja específica para la escritura de los macros. Abrir este tipo de hoja es tan sencillo como indicar [Archivo] [Nuevo] y seleccionar hoja de macros.

Inicialmente, se quieren rellenar una serie de celdas de la hoja activa con los títulos de los campos que llevará la agenda. Para ello existe la instrucción *Fórmula* que básicamente se encarga de colocar el valor indicado en la celda activa. Conjugando esta instrucción con *Seleccionar,* que permite activar cualquier celda, se pueden realizar escrituras en toda la hoja de cálculo.

Al igual que en la mayoría de las instrucciones, *Seleccionar* permite actuar de diferentes maneras. Por un lado, se puede seleccionar una celda de forma absoluta con un parámetro del tipo "L1C1" que activaría la primera columna y línea. Se puede hacer también de forma relativa respecto a la actualmente activa. El parámetro "LC(1)" por ejemplo, selecciona la celda inmediatamente a la derecha de la actual. Por último permite hacerlo con un rango de celdas, dejando activa una celda concreta. Esto se haría con algo como "L1C1:L5C6";"L2C3".

Los parámetros de las funciones, pueden a su vez ser recogidos de otro lugar. Para ello es enormemente eficaz la utilización de nombres en la hoja que señalen, al igual que las etiquetas, los valores más importantes del programa independientemente de su posición en la misma.

La asignación de un nombre a una o varias celdas, al igual que se hace en las hojas de cálculo, sólo exige seleccionar la celda o celdas y elegir *[Fórmula] [Definir nombre...]*, si bien es aconsejable tener los nombres en un lugar visible de la hoja (a la derecha o arriba de las celdas elegidas). Si se hace así, se puede seleccionar todo conjunto de celdas + nombres y utilizar *[Fórmula]* y *[Crear nombres]*.

Los nombres no sólo sirven como etiquetas de valores constantes en la hoja de macros, sino que además permiten señalar fácilmente puntos del programa, y así obtener los resultados de una determinada función o lugares a los cuáles se quiere saltar. Una precaución: en ocasiones será necesario diferenciar entre la celda a la cual se está haciendo referencia y el valor guardado en la misma. Por defecto y en caso de admitirse las dos posiblidades como parámetro de una función, el nombre indica la referencia, y *Valref(Nombre)* dará el valor guardado en dicha referencia.

Ejemplos de las funciones comentadas se pueen observar en la macro del *listado 1*que se encuentra en el fichero LISTADOS.TXT, pero existen otras. Se ha creído conveniente por ello, mostrar un resumen de todas las utilizadas a lo largo de este artículo en la tabla 1.

No se considera necesario dar el formato de estas funciones ya que pueden verse desde el mismo Excel, cuando se indica *[Fórmula] [Pegar función]* con la opción "Pegar argumentos". Hay que puntualizar que en la mayoría de las funciones no son necesarios todos los

TABLA 2: ENTRADA DE DATOS

D_Datos_Cliente	Elemento_Tipo	Pos_x	Pos_y	Anchura	Altura	texto	valor_inicial	Comentarios
D_Caja_diálogo_1				594	176	Datos del Cliente	2	
	5	40	22			&Nombre:		
D_Nombre_1	6	121	18	447			Sr. Importante	
	5	34	46			&Dirección:		
D_Dirección_1	6	121	42	446			Avda. del programador nº1	
	5	36	70			&Provincia:		
D_Provincia_1	21	121	66	160	108	T_Provincias	32	
	5	288	68			&Teléfono:		
D_Teléfono_1	8	409	65	160			1110100	
D_Prefijo_1	5	361	68			(91)		
D_Aceptar_1	1	94	132	88		Aceptar		
D_Cancelar_1	2	385	131	88		Cancelar		



argumentos y al no incluirlos Excel los sustituye por los valores por defecto.

Y CON LAS PALABRAS MAGICAS...

Para conseguir que el listado funcione, al igual que en todos los que siguen, es necesario primero crear los nombres que se indican en la columna de Nombres, además de definir como comandos los nombres de las macros. En el caso de *M_Crea_Cabecera* se hace seleccionando la celda, aplicando *[Fórmula] [Definir nombre]* y marcando la opción de *Comando*. Es posible asignarle también al macro una tecla rápida, que provoque su ejecución al pulsar *CTRL + tecla*.

Se debe tener la precaución de no ejecutar la macro en la misma hoja en la cual se ha realizado la programación ya que las actuaciones podrían provocar la pérdida de instrucciones. Es muy aconsejable salvar el archivo antes de ejecutar ninguna macro.

Si todo se realizó correctamente, se debería obtener en la hoja activa el resultado mostrado en la figura 1.

EL PROGRAMA INTERACTIVO

El programa ya es capaz de realizar una función, pero realmente lo único que está haciendo es repetir una sucesión de teclas más o menos compleja.

Uno de los elementos que permite darle más potencia a las macros es la posibilidad de ejecutarse interaccionando con el usuario, es decir, de cuando en cuando el programa debe detenerse a esperar una especificación del mismo. Excel dispone de varias posibilidades para realizar estas entradas.

El siguiente macro, permite abrir una hoja existente como agenda, o creará una nueva según las decisiones del usuario.

Los macros tienen la posibilidad de ejecutarse interaccionando con el usuario

Una de las formas de usar un cuadro de diálogo es utilizar uno de los ya existentes, como es el de abrir un fichero. La instrucción pone en marcha el proceso de abrir el cuadro tal y como ocurriría si se pulsara *[Archivo] [Abrir]*. El cuadro además devuelve un valor que indica si el archivo fue abierto o no.

La segunda manera es mediante la función *Alerta*, que da la opción de personalizar parcialmente el cuadro de diálogo. Permite seleccionar el texto, y también si el cuadro será de los siguientes tipos: Información, pregunta o error.

El tercer método ya preparado por Excel es mediante la función Introducir, que amplia aún más las posibilidades, permitiendo un nombre de cuadro, un texto y hasta una selección del tipo de dato que deberá ser pedido al usuario.

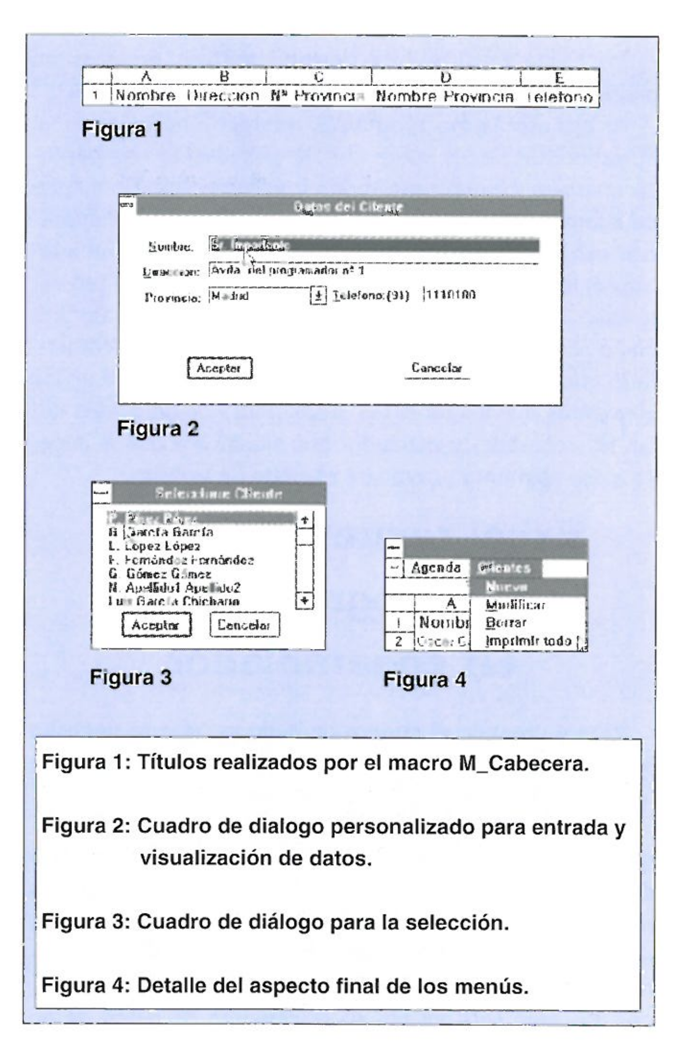


Figura 1: Títulos realizados por el macro M_Cabecera.

Figura 2: Cuadro de dialogo personalizado para entrada y visualización de datos.

Figura 3: Cuadro de diálogo para la selección.

Figura 4: Detalle del aspecto final de los menús.

TOMA DE DECISIONES

Pero cualquier lenguaje que permita entradas del usuario debe ser capaz de reaccionar a partir de ellas. Las macros disponen de instrucciones de decisión que son capaces de desviar el curso del programa. La más simple es *Si(Valor;A;B)* que en función de que el valor dado sea verdadero o falso, ejecuta A ó B respectivamente. Se tiene también *si....si.no....fin.si,* estructura que permite que A y B sean más de una instrucción, y el famoso *mientras.....salir.bucle* que, como es bien sabido, ejecuta el bucle una y otra vez hasta que la condición de mientras se consiga cumplir. Por último, el poco popular (en programación estructurada) *Ir.A* que provoca saltos a otros puntos del programa. Esta última instrucción, es conveniente utilizarla en combinación con etiquetas de nombre, si bien en cualquier caso, es considerada como una mala técnica de programación.

Estas instrucciones pueden verse en el *listado 2* de LISTADOS.TXT, que a su vez utiliza la macro preparada en el anterior ejemplo, como función dentro de éste.

La macro *M_Abrir_Archivo* obliga a abrir un archivo o crear uno nuevo con la cabecera antes indicada.

EL LIMITE ES LA IMAGINACION

Un nuevo paso a la hora de conseguir una interactividad "hecha a medida" llega de la mano del *Dialog*

Editor. Esta herramienta permite realizar cuadros de diálogo adecuados a nuestras necesidades concretas.

Al ejecutar dicho programa, aparece una caja vacía. Pulsando dos veces sobre la caja se presenta una pantalla que será común para todos los elementos. En ella se da información sobre la posición en la cual deberá aparecer, así como la altura y anchura del cuadro. Tiene además el texto que debe llevar asociado el elemento (en este caso la propia caja), el lugar donde se dará el valor inicial o el resultado de ese elemento concreto y un comentario que tan sólo sirve de documentación. Por último, los elementos pueden tener las propiedades de difuso (es decir, no accesible) y activador que significará que el acceso a ese elemento provocará el cierre de la caja.

Excel recuerda mucho a la programación en ensamblador

Para ir creando el cuadro de diálogo, sólo se necesita ir seleccionando en el menú la opción *[Elemento]* para luego ir rellenando con los diferentes elementos.

Para el programa se ha realizado un cuadro de diálogo personalizado que permite la entrada del nombre y dirección en formato de texto, la provincia, correspondiendo a una lista que se creará de todas las provincias españolas, el teléfono, que únicamente admite caracteres numéricos, y un elemento de texto (prefijo telefónico) que sólo será visible en la consulta de datos de un cliente. También se han incluído los botones de *Aceptar* y *Cancelar* típicos de cualquier cuadro de diálogo. El resultado cuando se ejecuta el cuadro de diálogo puede verse en un ejemplo en la figura 2.

Una vez dibujado el cuadro, se observa una cosa curiosa. El *Dialog Editor* no tiene capacidad para salvar los cuadros realizados de esta forma. Este hecho, que sorprende al principio, se ve explicado parcialmente por la forma en la que se ejecuta, como se verá a continuación. Para conseguir guardar un cuadro de diálogo, primero, *[Edición] [Seleccionar cuadro de diálogo] [Edición] [Copiar]*. Una vez se tienen los datos en el portapapeles se deben pegar en una hoja de macros de Excel (realmente se puede pegar en cualquier tipo de hoja, pero sólo así se podrá ejecutar). Realizando la operación inversa, se consigue tener en el *Dialog Editor*, el cuadro antes salvado.

En la tabla 2, está el cuadro correspondiente a la figura anterior. Los títulos son los nombres que por comodidad se han dado a los valores, ya que realmente, al pegar el cuadro en la hoja sólo se obtienen números y textos aparentemente inconexos. El *Elemento_tipo* señala en cada caso, el tipo de objeto. Respecto a los demás elementos corresponden directamente con los indicados en la caja de diálogo. Estos valores son perfectamente modificables en la hoja y en muchos casos se utiliza esta habilidad para ajustar las posiciones o variar los textos dinámicamente.

El carácter "&" que se observa en la tabla hace que el siguiente carácter corresponda con el subrayado. Como siempre en Ms-Windows, permite pasar a ese campo pulsando *ALT + carácter*. Otra característica interesante es que se pueden cambiar los textos correspondientes a los botones *Aceptar* y *Cancelar*.

Otra cosa: para los tipos de lista, el texto corresponde con el nombre de la lista que se pretende hacer aparecer. Así, este cuadro no funcionará hasta que no se defina una lista llamada *T_Provincias*. Dicha tabla junto con los prefijos, que se utilizará más tarde, está abreviada en la tabla 3 y forma parte también de la hoja de macros. Para crear una lista, al igual que con los nombres, se selecciona toda la tabla, incluídos los títulos, para después hacer *[Fórmula]* y *[Crear nombres]*. De igual forma se debe seleccionar todo el cuadro de diálogo (incluídos los títulos) y crear nombres. Esto último permite acceder a cualquier valor de forma sencilla.

Con la instrucción *Cuadro.De.Dialogo(D_Datos_Cliente)* el cuadro debe funcionar correctamente. Los "problemillas" que quedan, como son que se tengan limpias las entradas cuando se llame al cuadro (salvo la provincia), obligar a que se rellene el campo de nombre y pasar los datos a la hoja son algo que no requiere más explicación que la indicada en el *listado 3* del archivo LISTADOS.TXT.

Conocer el sentido de los valores *Elemento_Tipo* del cuadro de diálogo es de utilidad, ya que pueden variarse dinámicamente. Existen hasta 24 tipos diferentes de elementos y se numeran consecutivamente a partir de uno.

Alguno tiene características especiales; por ejemplo, que el texto en un tipo 6 no sirve para nada, o que el texto de una lista corresponde realmente con el nombre de dicha lista. Pero el número de elemento es el que realmente regula el objeto. Se debe saber que la condición de activador sólo consiste en sumar cien al número de elemento, y que mediante él se puede por ejemplo

TABLA 4: SELECCION DE CLIENTES

D_Selección_Cliente							
D_Caja_diálogo_2				294	126	Seleccione Cliente	1
D_Lista_Clientes_2	15	16	6	263	84	AGENDA2.XLS!Lista_Clientes	1
D_Aceptar_2	1	38	94	88		Aceptar	
D_Cancelar_2	2	148	94	88		Cancelar	



TABLA 5: ORGANIZACION DE LOS MENUS

Nombre Comando:	Nombre del comando visto por el usuario
Nombre Macro:	Macro que se debe ejecutar al seleccionar la opción
Sólo Macintosh:	Sólo sirve para asegurar la compatibilidad con dicho ordenador
Mensaje Estado:	Mensaje de ayuda que aparece en la barra de estado al seleccionar la opción
Ayuda:	Para identificar un fichero de ayuda personalizado

anular el botón de *Aceptar* poniendo el teléfono como activador. Al introducir el número el cuadro finalizaría inmediatamente sin necesidad de pulsar nada más. Por otro lado y como se verá más adelante en la opción de *Consultar,* se puede hacer que un campo no sea accesible, sumando doscientos al *Elemento_Tipo.*

Haciendo desaparecer elementos dinámicamente se amolda el mismo cuadro a lo que el usuario está pidiendo en ese momento.

EL PROGRAMA POLIMORFICO

Cuando se programa con macros de Excel hay que detenerse a pensar en algunas cualidades específicas. Por un lado, el lenguaje no es compilado, sino interpretado y el intérprete no es otro que el propio Excel. La orden *Establecer.Valor* sirve para modificar celdas en la misma hoja de macros. Un ejemplo de dicha actuación es la que se ha comentado respecto a los cuadros de diálogos pero no es la única. En un programa realizado con este tipo de lenguaje todo son variables. Cada celda de la hoja de cálculo es una variable ya definida y sobre la que se pueden lograr modificaciones. Las celdas que contienen el programa son por tanto material moldeable para sí mismo. En cierto modo, Excel visto de esta maner, recuerda mucho a la programación en ensamblador (aunque desde luego, no por su potencia). Al igual que en dicho lenguaje, la actuación se va a centrar directamente sobre su propio entorno (las hojas), si bien, también se puede salir más allá y realizar actuaciones sobre otras aplicaciones. Las funciones complejas recuerdan a las llamadas al DOS o a la BIOS.

Este símil, puede hacer ver más claramente la filosofía de este lenguaje.

Si se toman las celdas como las variables del programa se pueden considerar los nombres como punteros a dichas variables. Al igual que ellos, los nombres son modificables mediante las macros, y en este caso, se usarán para un nuevo cuadro de diálogo. Con la filosofía con la que se está enfocando el tema, un cuadro de diálogo no es más que una función compleja que recibe una serie de variables para su ejecución. En el caso de las listas, no se está indicando realmente una variable sino el puntero a dicha variable. Por ello, modificando las características del puntero se varía también la lista.

Se ha realizado el programa de selección de los elementos que se han ido introduciendo mediante el macro anterior. Una vez realizado el cuadro de diálogo de la figura 3 el programa necesita modificar la lista por dos razones: la primera, para que la lista se adapte exactamente a los nombres introducidos. La segunda es que el nombre es parte realmente de la hoja de cálculo y no de macros. Para ello, se crea su nombre a partir de varios trozos de texto (Técnica jamás utilizable en ningún lenguaje normal). Finalmente se llama al cuadro de diálogo.

Obviamente este pequeño programa será utilizado por varias rutinas. La orden *Detener,* sorprende un poco también, sobre todo a los amantes del lenguaje estructurado. Con esa línea se detienen todos los macros que hayan llamado a esta función, pero sin embargo no se sale del todo del programa (se verá posteriormente).

La función se puede ver en el *listado 4* en LISTADOS.TXT, que no tiene sentido si no es en conjunción con la tabla 4 referente al cuadro de diálogo.

Puede sorprender el hecho de utilizar el nombre "texto" o "valor_inicial" en esta nueva tabla que no tiene realmente estos títulos.

El Dialog Editor permite realizar cuadros de diálogo adecuados a nuestras necesidades

El truco utilizado en esta ocasión es una buena forma de guardar un grupo de cuadros de diálogo. Ya que la estructura en las columnas es siempre la misma, si se ponen los cuadros unos debajo de otros, se definen los rangos de los nombres (Pos_x, ancho_y, ...) como toda la columna a la cual corresponde. Se evita así una multiplicidad de nombres que provocaría confusión en el programa.

AGENDA VERSION 1.0

Aún con todo lo que se ha personalizado el programa hasta ahora, no se puede negar que se sigue en Excel. Llamar a los macros con *[Macro] [Ejecutar]* no es sencillo, y además existen muchas partes de Excel que no van a interesar al usuario.

Personalizar el menú, así como la barra de herramientas son los últimos pasos antes de tener una aplicación que merezca dicho nombre.

La agenda en esta primera versión tiene un menú como el mostrado en la figura 4. La barra de herramientas ha desaparecido, ya que no parece tener sentido, y se han añadido las funciones de borrar y modificar que no requieren ninguna explicación aparte de sus propios comentarios del *listado 5*, que se puede encontrar en el fichero LISTADOS.TXT, ya que únicamente vuelven sobre elementos ya tratados.

Además, para arrancar el programa, al abrir la hoja de macros, llamado AGENDA.XLM, automáticamente se comienza a ejecutar sin necesidad de un mayor control. De esta forma se puede incluso crear un nuevo icono en Ms-Windows que provoque la apertura de la agenda desde el mismo Ms-Windows.

El usuario final, se encontrará entonces con un icono a partir del cual, es llevado hasta un entorno de agenda creado a su medida.

Por otro lado, no se ha querido ocultar la hoja de cálculo aunque sería posible, de cara a que un usuario avanzado pueda manejar los datos perfectamente desde el Excel normal.

Para realizar el menú personalizado, se puede optar por añadir elementos a la barra existente, o crear una completamente nueva. Cada barra tiene un número a partir del cual poder ser llamada. Por otro lado existe otra orden para añadir un nuevo menú a partir del número anterior.

Agregar.Menu necesita como segundo argumento una tabla con una serie de columnas que contienen lo señalado en la tabla 5. El *listado 6* del fichero, junto con la tabla 6 es la parte de la agenda que personaliza el Excel para nuestra aplicación.

Un último apunte en cuanto a la tabla 6. Realmente para Excel son dos tablas independientes y por ello hay que definir los nombres Título 1 y Título 2 como separados para cada una de las tablas.

Se ha aprovechado también para mostrar un ejemplo del único tipo de bucle que no se había definido. *Para...salir.bucle()* actúa como la típica estructura *for...to* y que se ha utilizado para ocultar cualquiera de las barras de herramientas estándar del Excel.

Otra novedad, que se verá profundamente más adelante, corresponde con el uso de interrupciones. En Excel se pueden definir sucesos que provoquen el arranque de los macros. La estructura *Llamar.Por.Tecla* es la primera de este tipo en el programa. Al pulsar la tecla F1 se podrá volver a la hoja de macros para seguir trabajando con el menú habitual de Excel. Esta precaución se toma porque un macro que tiene el nombre *Auto_Abrir_....* se ejecuta automáticamente nada más abrir la hoja. Esto podría llegar a producir que no hubiera acceso al programa en un momento dado.

TRUCOS UTILES

Una de las cosas que hace más amigable la programación en este entorno son las herramientas que pueden aumentar la velocidad del trabajo. Por un lado, al ejecutar un macro se puede ver la opción paso a paso, que proporciona la oportunidad de utilizarlo como depurador del programa.

El usuario final, es llevado hasta un entorno de agenda creado a su medida

La búsqueda de errores con este método es completamente recomendable e ineludible su utilización para asegurarse de que la macro funciona correctamente, ya que no pueden ocurrir ningún tipo de fallos cuando se utilice la aplicación en algo serio.

Por otro lado, cuando se quiere realizar una acción con el programa y no se conoce muy bien con qué funciones se consigue, se puede poner *[Macro] [Grabar]* para luego ejecutar dicha acción. Tras eso, con *[Macro] [Finalizar grabación]* se consigue tener una hoja de macros en Excel que va indicando, en lenguaje de macros, cuáles fueron las actuaciones realizadas. Programar de esa manera siempre que se pueda permite optimizar el tiempo de programación de forma inusitada (y sobre todo, sin errores).

TABLA 6: MENUS DE LA AGENDA

	Nombre Comando	Nombre macro	Sólo Macintosh	Mensaje Estado
Título1				
	&Agenda			Operaciones sobre la aplicación
	&Salir	agenda.XLM!M_Salir		Salir del programa
Título2				
	&Clientes			Añadir, editar y eliminar clientes
	&Nuevo	agenda.XLM!M_entradaC1		Añadir un nuevo cliente
	&Modificar	agenda.xlm!M_ModificaciónC1		Modificar un cliente
	&Borrar	agenda.xlm!M_Borrar_cliente		Eliminar un cliente existente

PROGRAMAR BAJO MS-DOS EN 32 BITS

Agustín Guillén



Intel ha introducido por fin en los últimos procesadores un nuevo modo de trabajo que rompe definitivamente las limitaciones del 8086.

Tradicionalmente todas las aplicaciones bajo MS-DOS funcionaban bajo el clásico método de la segmentación de 64K y 16 bits, heredado de los antiguos procesadores de Intel: 8086, 8088, 80186 y 80188. Después vino el 80286, con su revolucionario Modo Protegido, similar en filosofía y casi totalmente compatible con el que poseen los actuales microprocesadores de Intel. Además no renunciaba al antiguo modo de funcionamiento de los "micros" previos, implementando el Modo Real, que otorgaba una total retrocompatibilidad. Y por último aparecieron los modernos 80386, 80486 y superiores, que mejoraban el Modo Protegido del 80286 e implementaban nuevas y más potentes características de cara a ponerlos al día en el competitivo mercado de los microprocesadores.

Los microprocesadores actuales dejaron la lista de modos de procesamiento de la siguiente manera:

1.- *Modo Real.16 bits.*
Es el modo de arranque del procesador (por compatibilidad).
Similar a un rapidísimo 8086, pero con alguna instrucción nueva y con los registros ampliados.

2.- *Modo Virtual de 8086 (V86)*
16 bits.
El procesador opera de manera similar a cuando está en Modo Protegido, quedando disponibles sus funciones, como la paginación. Al fin y al cabo es un Modo Protegido corriendo en el Nivel de Privilegio 3 (ver más adelante).
Los Registros de Segmento se utilizan de la misma manera que en Modo Real: con direccionamiento por Segmentos, no por Selectores.
El cambio a Modo Protegidó y viceversa es muy rápido. Permite tener varios "PC Virtuales" (V86) y procesos en Modo Protegido simultáneamente (multitarea).Más lento que el Modo Real.

3.- *Modo Protegido.32 bits.*
Modo natural del procesador.
Permite utilizar todas las características e instrucciones existentes.

Este artículo se va a centrar en este último modo y en todas sus características, tales como protección, depuración, memoria virtual, multitarea y en sus nuevas formas de direccionamiento de memoria. Después de un reset el micro arranca en Modo Real y con la paginación desactivada, por lo que se debe cambiar al Modo Protegido mediante software. Para activar dicho modo, se deben seguir ciertos pasos de cara a preparar el sistema: crear como mínimo la Tabla Global de Descriptores (GDT) con dos Descriptores, uno para un segmento de código y otro para un segmento de datos (además de cargar el registro GDTR con los datos del GDT); activar el bit PE del registro CR0 (activar el Modo Protegido propiamente); ejecutar una instrucción del tipo JMP (para limpiar las instrucciones ya leídas por el procesador y obligarle a decodificar de nuevo) y cargar los registros de segmento (todavia permanecen los del Modo Real). Hay que tener en cuenta que comenzará en el Nivel de Privilegio 0.

DIRECCIONAMIENTO DE MEMORIA

Existen dos modos de direccionamiento en Modo Protegido: la seg-

mentación y la paginación, aunque se pueden utilizar simultáneamente.

Segmentación.- Al igual que en Modo Real, el Modo Protegido utiliza dos valores de cara a formar la dirección lógica final: un Selector de 16 bits que se utiliza para averiguar la dirección base de un segmento, y un offset de 32 bits que se suma al inicio del segmento, para obtener al fin una dirección lineal de 32 bits. Esto nos permite direccionar en este modo 4 gigabytes. La segmentación es una de las bases de la protección, pues los segmentos son utilizados para encapsular zonas de memoria que contengan código o datos comunes. Toda la información referente a un segmento está recogida en una estructura de 8 bytes llamada Descriptor, y todos los descriptores del sistema están en tablas (Descriptor Tables), reconocidas directamente por el hardware.

Existen tres tipos de Tablas de Descriptores:

Global Descriptor Table (GDT): Contiene los descriptores que deberían ser accesibles por todos los procesos del sistema. La GDT puede contener cualquier tipo de descriptores, excepto los utilizados para las interrupciones.

Local Descriptor Table (LDT): Contiene los descriptores que están asociados a un determinado proceso, por lo que un segmento no puede ser accedido por un proceso si su descriptor no está ni en el actual LDT ni en el GDT.

Interrupt Descriptor Table (IDT): Contiene los descriptores que apuntan a la posición de las rutinas de las 256 interrupciones del sistema, su tamaño mínimo debe ser al menos de 256 bytes, para mantener los descriptores de las 32 interrupciones reservadas por Intel.

Cualquiera de estas tablas son de tamaño variable y pueden oscilar entre 8 bytes y 64 Kbytes (8192 descriptores). Cada una de estas tablas tiene un registro del procesador asociado a ellas: GDTR, LDTR y IDTR, que guarda la dirección base de 32 bits y su límite (tamaño) de 16 bits.

El modo natural del procesador y donde se pueden utilizar todas sus características e instrucciones es el protegido

Ya se han definido las Tablas de Descriptores, pero aún no a los propios descriptores, los cuales consisten en 8 bytes que contienen los atributos de cada segmento. Principalmente existen dos tipos de segmentos: segmentos del sistema y los que no lo son. El bit S en el descriptor lo determina.

En la figuras 1 y 2 se puede ver el formato general de los dos tipos de descriptores.

Los segmentos del sistema pueden ser de alguno de los siguientes tipos: Call Gate, Trap Gate, Interrupt Gate, TSS (Task State Segment), o cualquiera de ellos pero para el 80286 (compatibilidad).

Los segmentos locales (no del sistema) pueden ser: segmento de código, segmento de datos o segmento de pila (stack).

Paginación.- Este modo de direccionar memoria es muy útil para sistemas multitarea con memoria virtual. A diferencia de la segmentación que acota la memoria en "pedazos" de longitud variable, la paginación divide a los programas en múltiples Páginas de longitud fija: 4 Kbytes.

Este modo llega a poder direccionar hasta 64 terabytes, prácticamente memoria ilimitada.

Si se tuvieran que mapear todas las páginas en una sola tabla en memoria, ocuparía 4 Mbytes, por lo que se recurre al "truco" de tener dos niveles de tablas, antes de acceder a la Página final.

Cuando la paginación está activa, cualquier dirección lineal de 32 bits que se obtenga (incluso si viene ya calculada de la segmentación), se descompone en 3 partes: dos índices de 10 bits cada uno (Bloque de Directorio y Bloque de Tabla) y un offset final de 12 bits (Bloque de Offset). En la figura 3 se ven los pasos que sigue el procesador para convertir la dirección lineal a la física: el mecanismo de paginación utiliza el Bloque de Directorio y el registro CR3 (dirección física del Directorio de Páginas activo), para acceder en el Directorio de Páginas y obtener la dirección de una Tabla de Páginas determinada, después y mediante el Bloque de Tabla, se accede a la entrada de la Tabla de Páginas que contiene a su vez, la dirección física de la Página final y mediante el Bloque de Offset, se obtendrá la dirección física final en cuestión.

Como el tamaño de los Directorios de Páginas y de las Tablas de Páginas es de 4K, pueden albergar 1024 entradas cada una, por lo que por cada Directorio de Páginas, podemos acceder a 1024 Tablas de Paginas y por cada tabla a 1024 Páginas de 4K. Esto nos lleva a poder direccionar la memoria lineal completa del 80386 (4 gigabytes) con solo un Directorio de Páginas.

Cada entrada en las Tablas de Páginas contiene, ademas de la dirección de la Página (20 bits), informaciones referentes a dicha Página, tales

LISTADO 1. COMO SE COMPILAN LOS LISTADOS

```
tasm.exe /ml /m2 /q pm24sh
tasm.exe /ml /m2 /q ejemplo

tlink.exe /3 /x pm24sh pmode ejemplo, ejemplo
```

como si está presente en memoria, si ha sido modificada o simplemente leída, niveles de acceso o información para realizar el Caché a esa Página.

MEMORIA VIRTUAL

La Memoria Virtual hace posible la existencia de programas que manejen grandes cantidades de datos o la ejecución de infinidad de procesos simultáneamente. Se basa en el intercambio de páginas o segmentos (se suele utilizar la paginación por el tamaño fijo y discreto de sus elementos) entre la memoria y el disco duro. Cuando el procesador convierte una memoria lineal a física e intenta acceder a una Página que no se encuentra en memoria, genera una interrupción de fallo de Página y ejecuta una rutina para el tratamiento de páginas, que se encarga de cargarla de disco, marcarla como presente y no-sucia (no ha sido modificada) y devolver el control a la instrucción que generó la excepción.

PROTECCION

Los nuevos micros de Intel, incorporan ya todos los mecanismos para ofrecer una buena protección entre los distintos procesos y entre estos y el sistema operativo.

Incorpora cuatro niveles de protección (PL) que van de 0 a 3, siendo el 0 el más privilegiado y el que permite el acceso completo al sistema. Las reglas de privilegio son sencillas: sólo se podrá acceder a un segmento de datos si la aplicación tiene el mismo nivel o superior (PL más bajo) y, de la misma manera, solo se podrá ejecutar un segmento de código si este tiene el mismo nivel o inferior que el segmento actual. Hay que notar que el nivel activo es el que tenga el segmento de código que se esté ejecutando. Para aumentar el Nivel de Privilegio hay que recurrir a las Gates.

Con los cuatro niveles de protección se puede diseñar un sistema totalmente resistente a errores, estable y estructurado. Se puede colocar en el centro (nivel 0) al núcleo del sistema operativo; alrededor (nivel 1) a los servicios del sistema tales como los controladores de dispositivos, que deban modificarse a menudo; encima vendrían los OEM (Original Equipment Manufacturers), a los que les quedaría cierta protección contra los usuarios finales; y sobre todo ello, correrían las aplicaciones.

Con los cuatro niveles de protección se puede diseñar un sistema totalmente resistente a errores

A nivel de Paginación también está implementada la protección pero sólo a dos niveles posibles: usuario (accesible desde el nivel 3) o supervisor (sólo desde los niveles 0, 1 y 2), además de poderse activar cada Página como lectura o lectura y escritura.

Como último sistema de protección están las Instrucciones Privilegiadas, que solo pueden ejecutarse cuando el CPL es 0, ya que si no es así se generará una excepción de protección general. Son las siguientes:

La paginación es un modo de direccionar memoria muy útil para sistemas multitarea con memoria virtual

Las que modifican los registros de descriptores, de procesos, de depuración, de control o de testeo.

Las de acceso y control a la caché del micro.

La que bloquea el procesador: HLT.

DEPURACION

El propio hardware del micro provee de características de depuración (debugging) muy avanzadas. Se trabaja a tres niveles:

Con la instrucción INT 3 (optimizada a solo un byte: 0CCh), se genera una interrupción de depuración o breakpoint.

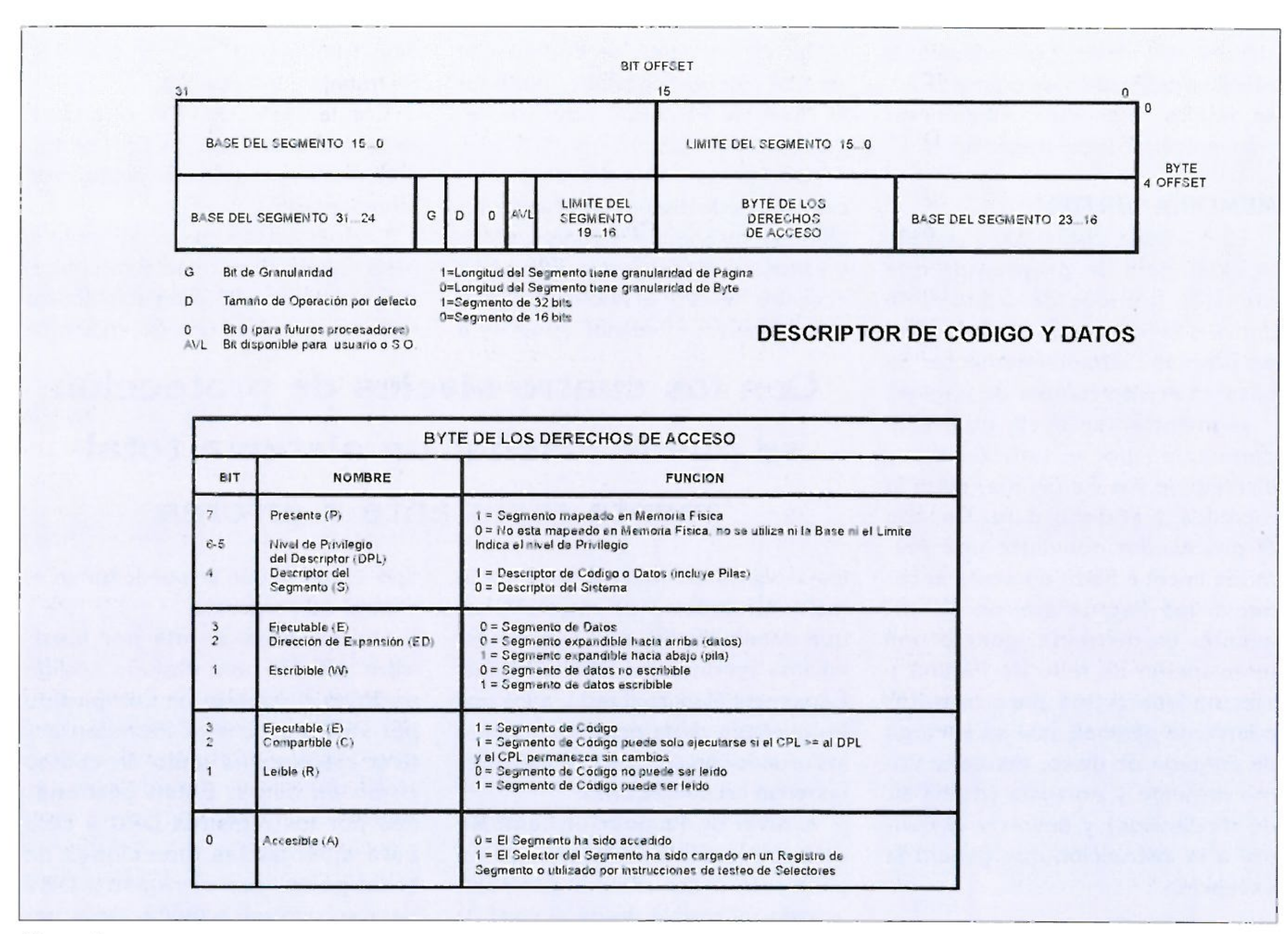
Posibilidad de ejecución paso a paso (single-step), mediante el bit 8 del registro EFLAG. Con solo activar este bit, se genera una excepción de tipo 1, con la que se puede tomar el control del programa.

Cuatro breakpoints por hardware, ideales para depurar código en ROM o cuando es compartido por varios procesos. Permiten activar breakpoints tanto en código como en datos. Están gestionados por los registros DR0 a DR3 para albergar las direcciones de breakpoint, por el registro DR7 para su control: habilitación y desactivación, tamaño o cuándo se activará (lectura, escritura o cualquier tipo de acceso) y por el registro DR6, que informa de las características en las que ha ocurrido el breakpoint: número y de qué tipo. Estos cuatro breakpoints por hardware también generan la excepción 1.

MULTIPROCESO

También están implementadas en el hardware diversas funciones para la multitarea. Existen dos tipos de descriptores relacionados con los procesos: los TSS y los Task Gates. Cuando la ejecución es transferida por alguno de estos descriptores, se ocasiona un Intercambio de Proceso. Es similar a una llamada a una subrutina, pero se salva mucha más información sobre el estado del procesador en una estructura en memoria llamada: Segmento del Estado del Proceso (TSS).

En el soporte de la multitarea, se involucran los siguientes elementos del sistema: TSS, descriptores TSS,



BYTE DE LOS DERECHOS DE ACCESO

BIT	NOMBRE	FUNCION
7	Presente (P)	1 = Segmento mapeado en Memoria Física 0 = No esta mapeado en Memoria Física, no se utiliza ni la Base ni el Limite
6-5	Nivel de Privilegio del Descriptor (DPL)	Indica el nivel de Privilegio
4	Descriptor del Segmento (S)	1 = Descriptor de Codigo o Datos (incluye Pilas) 0 = Descriptor del Sistema
3	Ejecutable (E)	0 = Segmento de Datos
2	Dirección de Expansión (ED)	0 = Segmento expandible hacia arriba (datos) 1 = Segmento expandible hacia abajo (pila)
1	Escribible (W)	0 = Segmento de datos no escribible 1 = Segmento de datos escribible
3	Ejecutable (E)	1 = Segmento de Codigo
2	Compartible (C)	1 = Segmento de Codigo puede solo ejecutarse si el CPL >= al DPL y el CPL permanezca sin cambios
1	Leible (R)	0 = Segmento de Codigo no puede ser leido 1 = Segmento de Codigo puede ser leido
0	Accesible (A)	0 = El Segmento ha sido accedido 1 = El Selector del Segmento ha sido cargado en un Registro de Segmento o utilizado por instrucciones de testeo de Selectores

Figura 1.

Registro de Procesos (Task Register) y descriptores Task Gate.

VIRTUAL 8086 (V86)

Existen dos modos para ejecutar código 8086 desde un 80386 o superior: en Modo Real o en Modo Virtual 8086. Este último permite acceder a parte de la potencia de los nuevos micros, como por ejemplo, ejecutar varias aplicaciones 8086, junto a otras de 32 bits, bajo un sistema operativo para 80386, e incluso con aplicaciones diseñadas para el 80286. Todo de manera concurrente. El modo de direccionamiento es idéntico al existente en Modo Real (segmentos de 64K), pero la paginación puede activarse para permitir ejecutar varios procesos 8086 simultáneamente y, aparentemente, en la misma zona de memoria. El hardware y el sistema se encargan de reubicarlos en cualquier punto de la memoria física, rompiendo la barrera de los 640K. Esto es totalmente transparente para los procesos 8086, aunque cada proceso puede utilizar hasta 256 páginas (el Mbyte máximo direccionable por el 8086) y cada una de ellas estar en cualquier punto de la memoria física (incluso en disco).

Las aplicaciones V86 corren con un CPL de 3, teniendo activados todos los sistemas de protección del Modo Protegido, por lo que cualquier intento de ejecutar instrucciones privilegiadas, generará una excepción 13. El Modo Real corre con un nivel de 0, anulando cualquier protección del sistema.

El manejo de interrupciones tampoco varía de una cara a las aplicaciones 8086 que están corriendo "emuladas", pero en realidad sí que se tratan de manera totalmente diferente. Todas las excepciones e interrupciones son "capturadas" por el micro (virtualizadas), y éste decide si devolverlas al sistema operativo 8086 o emularlas de manera conveniente. Del mismo modo todos los accesos directos a los puertos del sistema, mediante INs y OUTs, generados por aplicaciones no diseñadas para multitarea, son interceptados y gestionados por el micro, para que no existan conflictos entre distintos procesos. A esto se el denomina: Virtual I/O.

APIS 32 BITS

Son adiciones al sistema operativo, que se encargan o bien de dar acceso a habilidades propias del Modo Protegido, o para poder llamar a funciones del sistema (16 bits) una vez que está activado el Modo Protegido.

DPMI

Textualmente DOS Protected Mode Interface. Es el único API "casi estándar" completo, que da acceso al manejo de memoria, control total de interrupciones, excepciones, registros de depurado y emulación del coprocesador. Todo esto de manera elegante para que puedan

coexistir varios programas en modo protegido. Windows 3.1 incorpora (y ofrece a las aplicaciones que quieran utilizarlos) servicios DPMI 0.9, para comprobarlo solo hay que abrir una sesión de MS-DOS desde Windows y ejecutar cualquier buen programa de testeo de memoria, que nos informará de la existencia del sistema DPMI. A estos programas que ofrecen los servicios DPMI se les denomina Hosts.

VCPI

Virtual Control Program Interface. Es el predecesor del DPMI, y es una extensión del interface EMS, permitiendo a los programas DOS el ejecutarse en Modo Protegido, conviviendo con emuladores de Memoria Expandida o similares. Con el MS-DOS, se incluye un VCPI 1.0, sólo hay que tener instalado el dispositivo EMM386.EXE.

XMS

Extended Memory Specification. No es un verdadero API sino sólo unos servicios para manejar memoria extendida. Este servicio lo otorga el clásico HIMEM.SYS, y con el MS-DOS 6.2, se encuentra la versión 3.0 de XMS.

EXTENSORES PARA MS-DOS

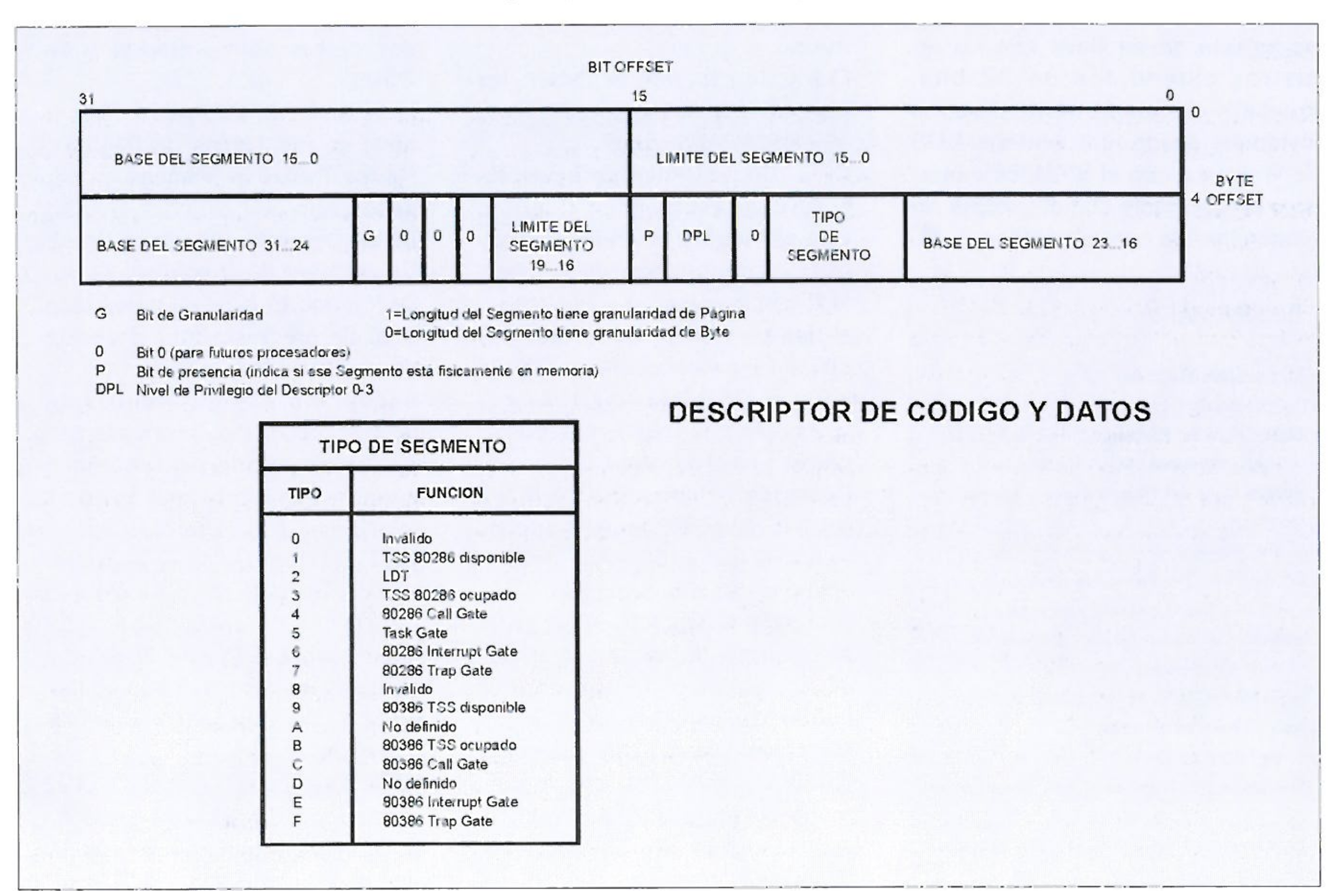
Ya se dispone de una gran variedad de extensores que otorgan la posibilidad de programar en 32 bits bajo MS-DOS, ofreciendo acceso a la totalidad del sistema operativo, algunos de los más utilizados son:

- 32-bit DOS/4GW DOS Extender de Rational Systems.
- Phar Lap 386/DOS Extender de Phar Lap Software, Inc.
- Ergo's OS/386 de Ergo Computing, Inc.
- PMode 3.02 desarrollado por Thomas Pytel (Tran), según declara ¡para su uso propio!.

Suelen funcionar bien bajo cualquier sistema, ya sea VCPI, DMPI, XMS o simplemente con el sistema "limpio" (modo raw). El código que da los recursos (el extensor propiamente dicho) puede unirse al código propio de diversas maneras: alguno ya va incluido en el ejecutable, enlazado previamente por el linker; otros se deben ejecutar previamente, antes de lanzar cualquier aplicación que se base en dicho extensor; y el resto va en un ejecutable externo, al cual llama el programa de 32 bits.

PMODE

En el disco adjunto se incluye la última versión del extensor para MS-DOS desarrollado por Thomas Pytel, llamado PMode. Permite a un programa DOS ejecutarse en Modo Protegido. Es igual si el sistema soporta DPMI, VCPI, XMS o está simplemente limpio (modo raw), ya que el extensor se encargará de crear los descriptores y tablas básicas del sistema y de dar soporte a un grupo de funciones compatibles con DPMI (manejo de descriptores, memoria extendida, control de interrupciones y ejecución de rutinas e interrupciones en Modo Real). En



TIPO DE SEGMENTO

TIPO	FUNCION
0	Inválido
1	TSS 80286 disponible
2	LDT
3	TSS 80286 ocupado
4	80286 Call Gate
5	Task Gate
6	80286 Interrupt Gate
7	80286 Trap Gate
8	Inválido
9	80386 TSS disponible
A	No definido
B	80386 TSS ocupado
C	80386 Call Gate
D	No definido
E	80386 Interrupt Gate
F	80386 Trap Gate

Figura 2.

caso de encontrarse el sistema ya previamente en DPMI, PMode solo hará de "puente" y pasará todas las llamadas al gestor DPMI. Como el DPMI está pensado para la convivencia de varias aplicaciones simultáneamente, todo el código se ejecuta con un CPL de 3; pero si se tiene el sistema limpio, con VCPI o con XMS, todo se ejecutará en el nivel 0 (mucho más rápido). Por otro lado, las llamadas al Modo Real se ejecutarán en Modo V86 si el sistema es VCPI o DPMI, o en Modo Real si se encuentra en XMS o limpio (raw). Por último, la paginación está desactivada en XMS o raw y activada bajo VCPI o DPMI.

Para ilustrar su manejo, se incluye un pequeño programa de ejemplo que informa del tipo de sistema actual (VCPI, DPMI, XMS o raw), así como de los bloques de memoria máximos disponibles, tanto en memoria convencional (por debajo del primer Mb), como en memoria extendida. Ademas ejecuta una llamada al DOS en Modo Real (lectura del teclado a traves de la BIOS) y alguna función desarrollada con los registros extendidos de 32 bits. Pruebe a ejecutarlo desde diversos sistemas: desde una ventana DOS de Windows, con el EMM386 cargado o simplemente con el sistema sin ningún tipo de controlador (raw). ■

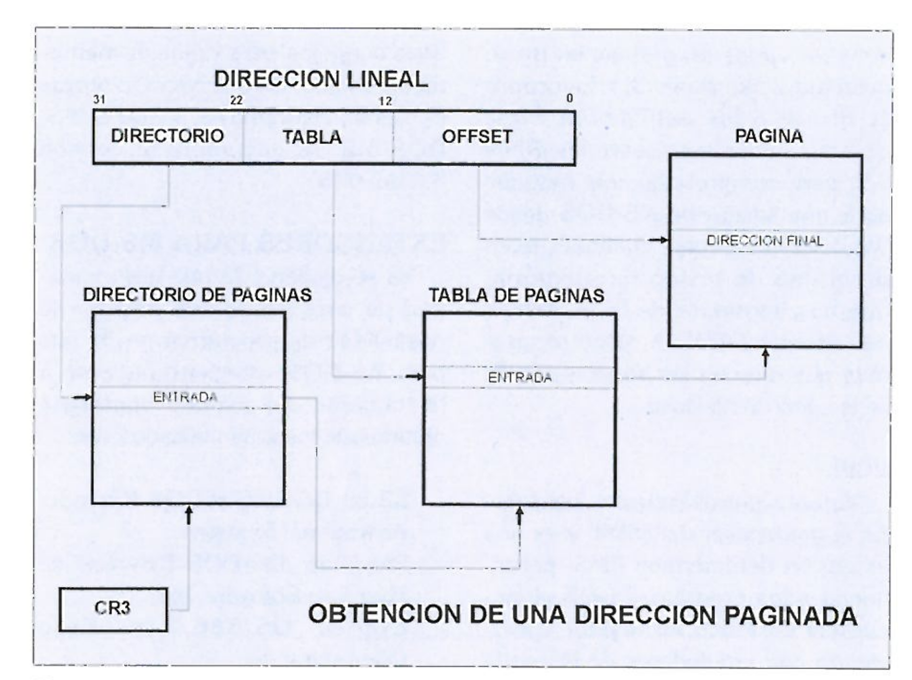


Figura 3.

BIBLIOGRAFIA

Introducción al 80386
Intel/Anaya Multimedia

80386 Guía del Programador de Sistemas
Intel Corporation/Anaya Multimedia

80386 Guía del Programador y Manual de Referencia
Intel Corporation/Anaya Multimedia

80386 Manual de referencia Hardware
Intel Corporation/Anaya Multimedia

Intel486 DX Microprocessor. Preliminary
Intel Corporation

Intel486 Microprocessor Family Programmer's Reference Manual
Intel Corporation

Dos Protected Mode Interface Specification Rev. 0.9

Dos Protected Mode Interface Specification Rev. 1.0

GLOSARIO

API.- Advanced Programing Interface. Se suele llamar así, de manera genérica, a cualquier interface para acceder al Sistema Operativo, o a ampliaciones del mismo.

Call Gate.- Un tipo de Descriptor para invocar a un procedimiento mediante CALL o JMP.

CPL.- Current Privilege Level. Es el Nivel de Privilegio en el que se está ejecutando el código actual. El Nivel de Privilegio es la "libertad" con la que se puede acceder al sistema y el nivel de protección aplicado a dicho código. Oscila entre 0 (máxima libertad) y 3 (más control y más lento). Se almacena en el registro CS.

Descriptor.- Estructura de 8 bytes que define el tipo de segmento, su dirección base, su límite y el tipo de acceso permitido.

Dirección Física.- El espacio de direcciones limitado por el bus del procesador. 32 bits para el 80386 (4 gigabytes).

Dirección Lineal.- Una dirección de 32 bits en un espacio de memoria sin segmentación. Si la paginación está activa, la Dirección Lineal se convertirá en Dirección Física. Si no hay paginación, la lineal y la física, coinciden.

Dirección Lógica.- La dirección con la que trabajan las aplicaciones y programas. Consiste en dos partes: un Segmento y un Offset.

Directorio de Páginas.- Primer nivel de las Tablas de Páginas. Mapea Tablas de Páginas en vez de Páginas.

DPL.- Descriptor Privilege Level. Es el Nivel de Privilegio de un Descriptor. El nivel de protección final de un Descriptor depende de su DPL y de su CPL.

Excepción.- Interrupción que ocurre cuando hay una violación de las reglas de protección o cuando se ejecuta una instrucción INT.

GDT.- Global Descriptor Table. Es una Tabla de Descriptores que contiene muchos tipos de Descriptores. Puede contener Descriptores del tipo TTS y del tipo LDT. Solo puede haber una en el sistema.

IDT.- Interrupt Descriptor Table. Tabla de Descriptores que contiene la información sobre el código de las 256 interrupciones.



Interrupt Gate.- Un Descriptor utilizado para invocar a una rutina de interrupción.
LDT.- Local Descriptor Table. La Tabla de Descriptores que suele contener los Descriptores de un proceso en particular. Cada proceso puede tener su propio LDT, pero solo puede existir un GDT en todo el sistema.
Memoria Extendida.- La memoria que está sobre el 1 Mb, solo puede ser accedida en Modo Protegido.
Memoria Expandida.- Es otra forma de acceder a la memoria que se encuentra sobre el primer Mb, consiste en paginar porciones de la memoria superior en una zona de memoria accesible por el micro en Modo Real, generalmente entre los 640K y los 1024K.
Memoria Virtual.- Memoria extra que se encuentra por encima de la memoria física del sistema. Realmente no existe más memoria, pero el hardware, el sistema operativo o el Host DPMI lo emula, a base de realizar intercambios entre las distintas zonas de memoria real y un dispositivo externo, generalmente el disco duro. De esta manera, aparentemente disponemos de sistemas con cientos de Mbytes de memoria, aunque muchísimo más lentos.
Offset.- Desplazamiento relativo a un punto determinado de la memoria o a un bit dentro de un Registro o byte.
Página.- Un bloque de memoria de 4K. El 80386 puede aplicar protecciones especiales tales como solo-escritura o Nivel de Privilegio a cualquier Página del sistema.
Paginación.- Forma de administrar memoria utilizada para simular un enorme y sin fragmentar espacio, usando una pequeña y fragmentada memoria y espacio en el disco duro.
Proceso.- Tarea o código que está ejecutando el procesador. Puede haber muchos procesos corriendo simultáneamente (multiproceso).
RPL.- Requested Privilege Level. Indica el Nivel de Privilegio para un determinado Selector. Está contenido en los dos bits menos significativos del Selector.
Segmentación.- Forma de administrar memoria que permite múltiples zonas de memoria independientes y protegidas entre si.
Segmento.- Bloque de memoria lineal. En Modo Real, los Segmentos son siempre de 64K y están limitados al primer Mb de memoria. En Modo Protegido, un Segmento puede empezar en cualquier lugar de la memoria y tener cualquier longitud (hasta 4 gigabytes).
Selector.- Indice a la Tabla de Descriptores, sólo en Modo Protegido. Los Selectores son usados en vez de los clásicos Segmentos en los Registros de Segmento.
Tabla de Páginas.- Tabla de 4K que contiene la información de 1024 páginas. Cada Tabla de Páginas mapea 4Mb de memoria lineal a memoria física.
Trap Gate.- Un Descriptor utilizado para llamar a una rutina de excepción.
TSS.- Task State Segment. Estructura de datos que se utiliza en el intercambio entre Procesos y para protección del sistema.
Virtual 8086 Mode (V86).- Modo de ejecución que permite la emulación con el 8086 y a la vez la multitarea.



FUNDAMENTOS DE UNA NUEVA INTELIGENCIA

Rafael A. Cazorla

Ahora que los ordenadores están aprendiendo a hablar, el paso siguiente hacia una máquina capaz de relacionarse con el mundo real, consiste en enseñarles a que reconozcan los objetos que le rodean.

Todo el mundo recordará la película "2001: Una Odisea en el Espacio" donde el ordenador central "HALL 9000" identificaba a los tripulantes de la nave, llegando incluso a leer de los labios de los astronautas. Dentro de la Inteligencia Artificial (IA) siempre se ha tenido por una premisa (para que una maquina sea "inteligente") la posesión de la facultad de observar el entorno en que se desarrolla y su posterior análisis de los objetos que le afectan.

APLICACIONES

El Reconocimiento de Objetos es uno de los principales proyectos de investigación a nivel mundial, tanto por sus increíbles aportaciones a la Ciencia, como es el caso del desarrollo de robots autónomos para la colonización de Marte, la industria, microcirugía, o en el campo de los sistemas de seguridad, el reconocimiento de personas para acceder a niveles de alta seguridad, donde una simple clave no era suficiente, y tantos otros proyectos a realizar que hasta hace poco sonaban a ciencia ficción y ahora comienzan a ser factibles gracias a este nuevo progreso.

Si se logrará una máquina capaz de ver y reconocer los objetos que están a su alcance, estaremos a punto de conseguir el sueño de todo informático: Un Robot Inteligente.

Actualmente hay establecida una carrera entre las grandes universidades (M.I.T, U.C.L.A.), y los gigantes de la industria informática, fomentada por las superpotencias industriales, para conseguir el sistema más eficiente, en rapidez y fiabilidad, de Reconocimiento de Objetos.

TECNICAS

Cualquier programa que tenga como función la identificación de las figuras incluidas en una imagen, seguirá un método distinto de procesamiento de la imagen, dependiendo de la finalidad que vaya a tener: Procesamiento bidimensional y en segundo lugar el Procesamiento tridimensional (o a veces "del mundo real").

El Procesamiento Bidimensional es empleado con asiduidad en líneas de ensamblaje automático, donde las figuras a reconocer no son muy complejas. Tiene por

Los ordenadores actualmente pueden "hablar" (emitir sonidos) y "oír" (reconocer la voz). También pueden imprimir o mostrar imágenes. Solamente les faltaba "ver". Las técnicas de reconocimiento de formas que emplean I. A. hacen posible que un PC distinga los objetos del mundo real.

objetivo el conseguir un modelo de 2D que se aproxime lo más posible a la realidad; para ello se realiza un filtrado de la imagen mediante un intensificador de contraste, lo que consigue un fácil y rápido método de obtener los contornos, convirtiendo la imagen dada (usualmente en color) a blanco y negro (imagen binaria), facilitando al ordenador reconocer los limites del cuerpo mediante algoritmos sencillos. Este método es eficiente en ambientes muy restringidos, ya que procesa las imágenes como si fuesen planas, por lo tanto puede causar problemas cuando se trata de analizar figuras tridimensionales, donde alguna puede estar parcialmente superpuesta a otra, provocando errores de interpretación. Tomemos por ejemplo un triángulo colocado casualmente encima de una figura cuadrada. Lo que la computadora supondría que esta viendo serian dos triángulos pegados por su base, formando un cuadrado.

El Reconocimiento de Objetos es uno de los principales proyectos de investigación a nivel mundial

Otro error característico de este tipo de análisis, es que si se toman fotos de objetos, como una pirámide, si la cámara le enfoca verticalmente creerá que esta mirando un cuadrado. Este tipo de cuestiones no los puede resolver el Procesamiento Bidimensional, lo que hace necesario el uso de la imagen tridimensional.

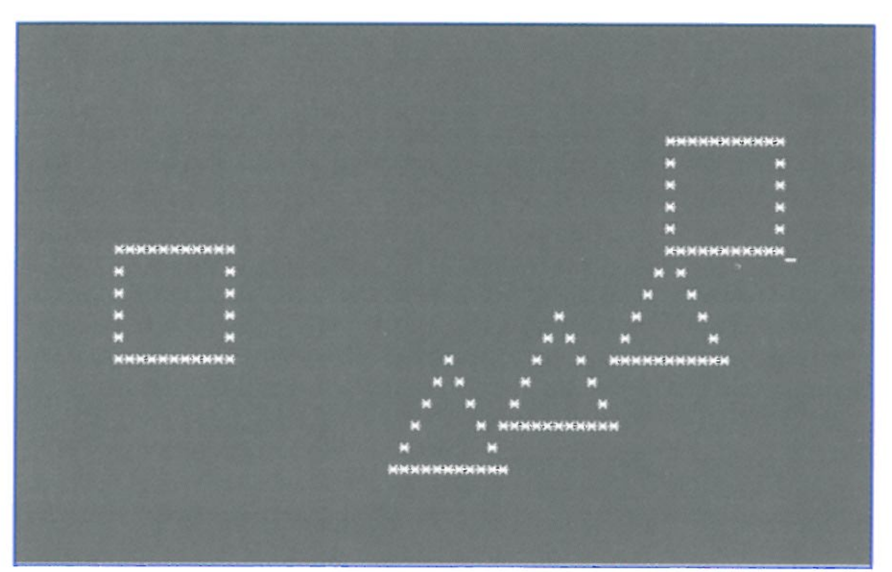
El Procesamiento tridimensional trata de alcanzar un modelo en 3D a partir de la imagen dada, intentando solucionar todos los problemas de visualización provocados por objetos que se superponen. Aquí es donde la "I.A." muestra su grán poder de resolución de problemas. Se exige el poder de decisión a la computadora, ya que hay problemas que deben ser resueltos antes de que un programador implemente un sistema de visión artificial. Aunque sólo se disponga de una imagen para analizar, es posible encontrar una representación bastante aproximada de la realidad. Ocurre un efecto semejante al que se experimenta al mirar con sólo un ojo: no se distinguen bien las distancias, es más, un bebé no sabe la distancia que hay entre las cosas que le rodea. El reconocimiento de las distancias entre objetos se consigue relacionando varios factores (color, sombras, brillos, texturas...) y fundamentalmente mediante la experiencia. Este es el trabajo de los programadores, y la Inteligencia Artificial es el método de reproducir en la computadora la forma en que las personas procesan las imágenes. Por lo tanto, no sólo le basta conocer los contornos, sino que aquí se aplican otras técnicas, como el estudio de los colores, brillos, profundidades, texturas. Pero este incremento de información tiene sus inconvenientes; se realiza un gasto de memoria considerable, en una imagen binaria cada pixel es almacenado en un bit (b/n), mientras que para la tridimensional es necesario conocer el brillo de cada pixel, el cual, dependiendo de la definición que estemos usando. Si se tienen 256 tonos de gris, requiriendo cada pixel 1 byte, es decir la imagen tridimensional es 8 veces mayor a la binaria.

Aunque con este procesamiento generalmente baste con una sola imagen, esto no siempre funciona y requerirá de una nueva imagen desde un nuevo ángulo para identificar al objeto, ya que puede estar oculto o simplemente que se encuentre en una posición en la que sea imposible reconocer a la pieza (si es que se busca una interpretación correcta de la realidad). Como este doble procesamiento puede ser demasiado lento, debido a que los algoritmos empleados en el reconocimiento de objetos suelen ser de orden de n x n , o incluso de órdenes mayores, generalmente se despreciará el análisis exacto de la realidad por un modelo que se aproxime lo suficiente como para que resulte un rendimiento aceptable.

Un factor importante en un programa que se denomine "inteligente" es que sea capaz de aprender, en este caso nuevas figuras, por lo que ante ciertas circunstancias donde no es posible identificar al objeto por sus cualidades físicas, almacena éste en la base de datos, caracterizándolo como una nueva figura. De esta manera el programa "aprende" ese nuevo objeto, de modo que ante otra posible ocurrencia lo identificará. Estos

```
Escribe las coordenadas (x y) del triangulo : 60 10
Mas triangulos? sEscribe las coordenadas (x y) del triangulo : 50 13
Mas triangulos? sEscribe las coordenadas (x y) del triangulo : 40 15
Mas triangulos? nEscribe las coordenadas (x y) del cuadrado : 10 10
Mas cuadrados? 6Escribe las coordenadas (x y) del cuadrado : 60 5
Mas cuadrados? _
```

El primer paso es situar los objetos en pantalla. El máximo es 25 para cada tipo.



El programa sitúa las figuras sólo con su contorno, pudiendo montarse una figura sobre otra para simular objetos y caras ocultas.

sistemas son empleados para la extracción de información topográfica de una imagen.

Si se quiere conocer la textura de los cuerpos hay que analizar sus apariencias. Para esto es imprescindible estudiar cómo reflejan la luz, puesto que lo harán de distinta forma si se trata de superficies lisas o rugosas. Normalmente se identificarán los cuerpos duros con superficies lisas y los suaves con las rugosas. Las materias lisas reflejan la luz de una forma uniforme, mientras que las rugosas, lo hacen dispersadamente. Hasta aquí muy bien, pero no se ha tenido en cuenta que en situaciones reales el brillo relativo a un cuerpo es alterado por su color y por las propiedades reflectantes de las materias de las que esta compuesto. Por lo tanto, se exigirá tomar dos o más imágenes, de los objetos desde diferentes posiciones, en relación con la fuente de luz. Esta característica es empleada en cadenas de ensamblaje de automóviles, para comprobar la uniformidad de la chapa y su posterior pintado.

UN CASO PRACTICO

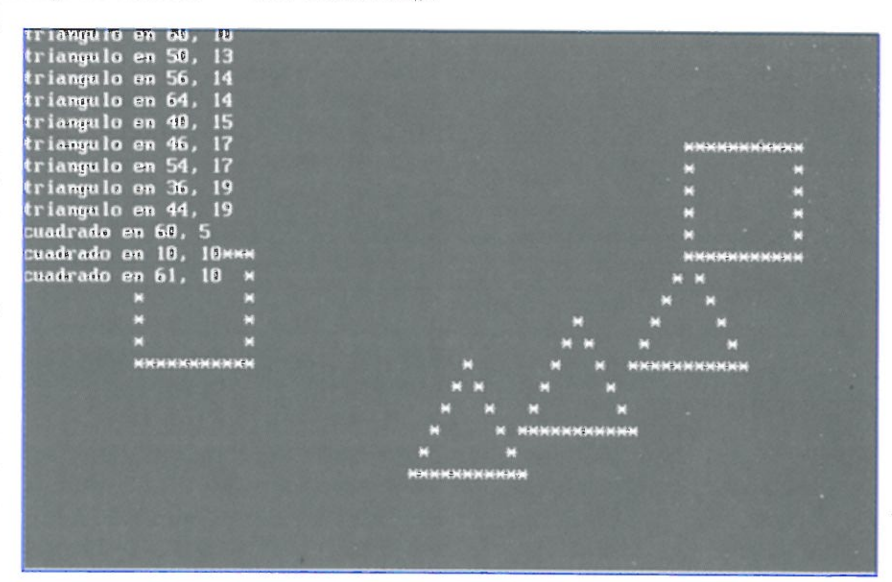
El programa que se comenta a continuación es un ejemplo típico de procesamiento bidimensional. Evidentemente este proceso sólo proporciona datos sobre figuras planas, perdiendo información si el objeto que estaba en la imagen era tridimensional, puesto que no se tiene en cuenta el factor 3D (sombras, brillos, superficies, texturas, colores, profundidad, etc). Algunas funciones que ya se han implementado gracias a esta técnica sirven para detectar fisuras o deformaciones en una pieza industrial o la clasificación de piezas en una cadena de montaje.

Este ejemplo supone que ya se han conseguido los contornos de la imagen, ya que el proceso de digitalización de la señal de una cámara de televisión no se realiza vía software, y la función que realiza es el análisis de figuras simples, como cuadrados y triángulos isósceles, mediante el estudio de los ángulos que crean las aristas a partir de un vértice dado.

Se han elegido estas dos figuras puesto que los algoritmos que las controlan y analizan son relativamente fáciles de comprender para alguien que conozca el lenguaje C. Se ha elegido este lenguaje, para realizar la implementación, puesto que es uno de los más populares, proporcionando un esqueleto estructurado que tiene como límite la capacidad creadora de cada programador, aportando una mayor eficiencia y rapidez ante problemas relativamente simples, pero que hubiesen costado mayor esfuerzo si se hubiese empleado Lisp u otro lenguaje propio de la IA.

Toda figura se puede analizar como una asociación de triángulos

Aunque los dos cuerpos son muy fáciles de reconocer, esto no implica que no tengan una mayor importancia, puesto que toda figura se puede analizar como una asociación de triángulos. La función que crea los triángulos es *triangulo()* y la de los cuadrados *cuadrado()*.



El proceso termina con la presentación de los puntos claves que determinan el tipo y posición en pantalla de los modelos.



El método de reconocimiento se basa en el análisis de los ángulos característicos que crean estas figuras. Sólo necesita comprobar dos puntos adyacentes a un vértice. Si el ángulo formado entre los dos puntos es de 60 grados, se tratará de un triángulo isósceles y si se trata de un ángulo recto será un cuadrado. Para esto simplemente se ha de comprobar la posición de estos dos puntos. Si es un triángulo, estos adyacentes se han de encontrar en las dos diagonales del vértice elegido, y como es lógico, en un cuadrado estos estarán uno en

La librería Rec_obj.h reúne las funciones de creación y análisis de los objetos que trata de reconocer el programa

la vertical y otro en la horizontal del punto de partida. Las funciones que se encargan de reconocer los cuerpos son: comprueba_triangulo y comprueba_cuadrado.

El programa consta de tres partes: la función principal ("Reconoce") y dos librerías: Csr_rac.h y Rec_obj.h.

La librería Csr_rac.h contiene las funciones que manejan el cursor y el buffer de pantalla ha sido implementada ya que aún existen compiladores de C que no las poseen (Microsoft C version 4). La imagen de video se simulará en forma de texto: constará de 24 x 80 pixels, así el código es totalmente portable a cualquier compilador.

La librería Rec_obj.h reúne las funciones de creación y análisis de los objetos (cuadrados y triángulos en esta versión) que trata de reconocer el programa. Se han añadido las funciones de creacion de las figuras para simular el filtrado que consigue al simplificar las figuras a sus contornos, puesto que ese aspecto se suele realizar a nivel de hardware.

En la función principal se encuentran las llamadas a las funciones de creación de los contornos de los dos tipos de figuras: cuadrados y triángulos. Hay un límite de veinticinco objetos como máximo por tipo. Después de pintarlos en pantalla el programa espera a que se le de a cualquier tecla para comenzar a analizarla, devolviendo los vértices que denominan a las figuras. Como efectúa un barrido de izquierda a derecha, presentará como vértices característicos de un cuadrado, los que se encuentren más a la izquierda del usuario. Estos puntos clave los escribirá en pantalla, pero si alguien cree necesario que se graben en un archivo, la modificación es muy simple, incluso sin necesidad de modificar el código, puesto que al utilizar la pantalla en modo texto (para facilitar la portabilidad del ejecutable) el usuario puede redireccionar el programa a un fichero.

Si se desea analizar la eficiencia de este, hay que verificar que se cumplen una serie de condiciones. La primera sería:

LISTADO

```
int CompruebaTriangulo(x,y)
/* Verifica si en las coordenadas dadas existe un triangulo
isósceles mediante a comprobación de los ángulos caracterís-
ticos, devolviendo VERDAD O FALSO dependiendo de su
existencia.
                    *
El angulo de 60 grados será:  *  *
*/
int x,y;
{
if(comprueba_punto(x+1,y+1) && comprueba_punto(x-1, y+1))
  return 1;
 return 0;
}

int CompruebaCuadrado(x,y)
/* Esta función verifica si en las coordenadas dadas existe un
cuadrado, devolviendo VERDAD O FALSO dependiendo de su
existencia.
                    * *
 El angulo de 90 grados será:  *
*/
int x,y;
{
 if(comprueba_punto(x+1,y) && comprueba_punto(x, y+1))
  return 1;
 return 0;
}
```

¿Afecta la superposición de objetos al reconocimiento de los mismos? No, el programa las detecta sin problemas puesto que al no tener en cuenta caras ocultas, los ángulos son respetados.

¿Afecta el tamaño? No, ya que con el método empleado en el reconocimiento de los objetos es indiferente el tamaño de estos.

Al tener un ámbito tan restringido de trabajo, el análisis resulta extremadamente preciso

¿ Es eficaz ? Sí, puesto que al tener un ámbito tan restringido de trabajo, el análisis resulta extremadamente preciso. Hay que tener en cuenta que sólo funciona con triángulos isósceles. Si se le hiciese trabajar con triángulos rectángulos el resultado fácilmente sería erróneo al identificarlo con un cuadrado, si es un vértice con el ángulo recto el primero que encuentra, en otro caso, podría acertar si es un angulo menor el que encuentra antes.

DETALLES TECNICOS

En el listado están las funciones que reconocen la figura en pantalla. Como se puede comprobar, han de estar situadas en una determinada posición, puesto que no se controla si estan dibujadas inclinadas, o con un cierto angulo. Por lo tanto, ante este suceso, problamente habría falsas interpretaciones de lo que hay representado. Si se desea profundizar en el tema, es recomendable desarrollar un editor de figuras donde se puedan realizar movimientos a las figuras para colocarlas en posturas distintas, y volver a ejecutar el programa, de manera que se observen los errores, y se implementen nuevas soluciones con los métodos ya expuestos.

Se han de unificar todos los métodos para que el ordenador seleccione el procesamiento más eficiente en cada caso

Para compilarlo se deberá añadir las librerías al directorio donde se encuentren las del compilador que se este utilizando, en el caso del Turbo C será el "include". Si se desea ejecutar debe poner en la línea de comandos del DOS "C:\>reconoce".

OTROS METODOS

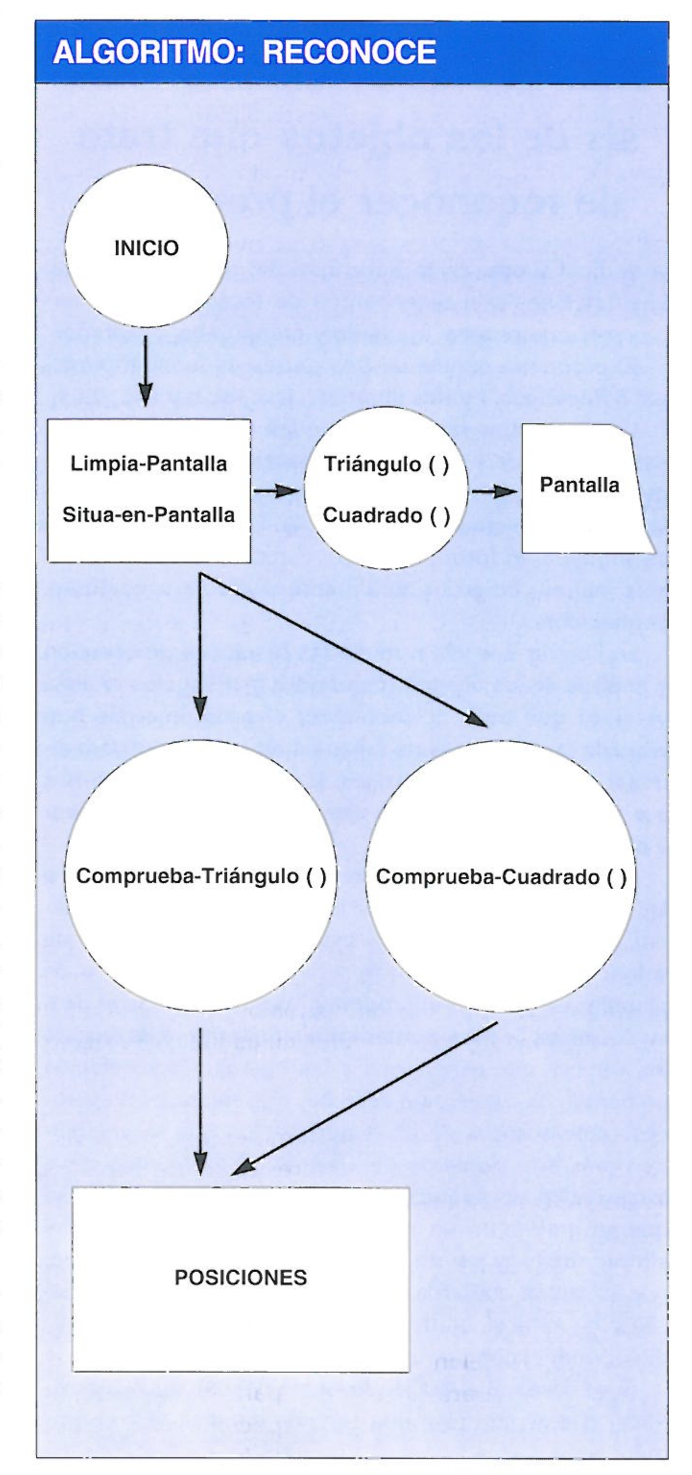
Existen otras técnicas de reconocimiento de estas figuras simples. Una de ellas es utilizar puntos clave de manera que sean característicos de cada figura. En el ejemplo del cuadrado los puntos claves serían los dos vértices adyacentes al de origen, en el triángulo isósceles bastaría con comprobar el vértice de la derecha, y si se añadierá un triángulo rectángulo se estudiaría igual que el cuadrado. Pero el método más extendido es mediante el reconocedor delta-D. El sistema consiste en estudiar el cambio de dirección de las aristas. Cada figura cambia de dirección un número característico de veces: un triángulo 3, cuadrado 4, pentágono 5,... incluso el círculo se puede estudiar como una figura que cambia infinitas veces. Este numero de cambio de direcciones se le denomina Delta (o Dirección). Como se puede imaginar, este proceso acarrea una mayor complejidad en el codigo, y aunque es un procedimiento más "inteligente", es independiente del tamaño del cuerpo y no le influye la posición del objeto a analizar. Pero tiene un gran fallo: posiblemente no consiga averiguar qué figuras hay en pantalla cuando se superponen varias.

Por tanto, si se desea construir un programa que posea una grán potencia de reconocimiento, se han de unificar todos los métodos, para que dependiendo de la situación la computadora seleccione el procesamiento más eficiente en cada caso. ■

BIBLIOGRAFIA:

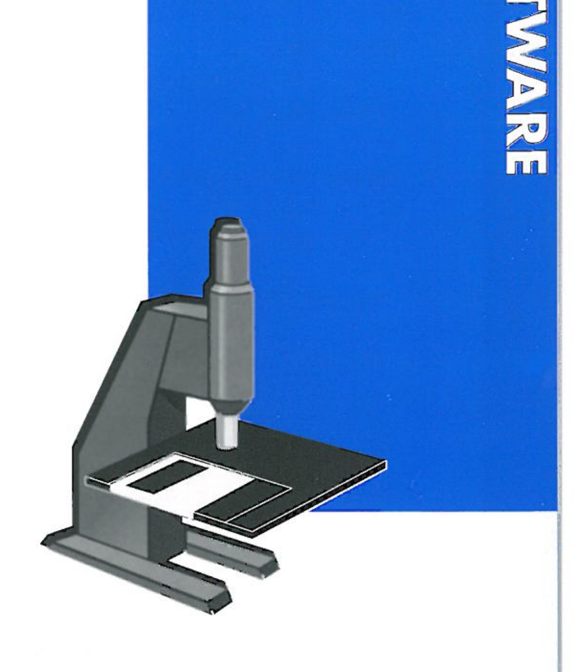
Advanced Graphics Programming in C & C++
por Roger T. Stevens y Christopher D. Watkins
Ed: M&T Publishing, Inc.
y
High-Resolution Computer Graphics Using C
por Ian O. Angell . Ed: Macmillan Education LTD.

ALGORITMO: RECONOCE



CONTROL DE LOS PROGRAMAS EN MEMORIA

Francisco Martín



LA IDEA DEL PROGRAMA

Este monitor de memoria está concebido como un diseño básico que deberá ampliarse de acuerdo a los gustos particulares de cada desarrollador.

Se ha escrito en ensamblador (en el fichero RESIMEM.ASM que generará RESIMEM.COM) únicamente por motivos de optimización de espacio. Ocupa menos código y datos que si se hubiese compilado por ejemplo en C o en Pascal, debido a que no incorpora las rutinas de runtime.

Por lo demás no es apreciable la ganancia en velocidad, de manera que si no se está preocupado por el espacio que pueda ocupar, las rutinas aquí usadas se pueden reemplazar fácilmente por las rutinas de librería incorporadas por el compilador de C, Pascal, etc; siendo aconsejable realizar el interfaz con bajo nivel en ensamblador ya que permite un mejor control.

Un ejemplo de cómo se puede realizar desde lenguaje C un programa equivalente se muestra en los archivos RESIM.C y RESI.ASM (este último realiza la interconexión de bajo nivel con RESIM.C. Juntos generarán RESIMEM.EXE). En él están implementadas todas las rutinas necesarias para acceder al MENU, dejando como una elección personal la implementación del resto. De hecho, éste se podría considerar como un esqueleto de propósito general.

El programa admite dos modos de funcionamiento: como un programa "normal" y como un programa residente. Esto dependerá de los parámetros pasados en la línea de órdenes. Opcionalmente admite hasta tres parámetros precedidos por el símbolo '/' o por '-', en mayúsculas o minúsculas indistintamente. Estos son:

R - Solicita cargar el programa en modo residente. Sólo se admite una copia residente en memoria (véase excepción más adelante).

D= - Usado para especificar la letra de la unidad de disco que se usará para preservar los datos de pantalla cuyo tamaño exceda el del buffer interno (se explica más adelante en detalle). Por ejemplo: "RESIMEM -d=c" usará la unidad C para almacenamiento. Si este parámetro no se especifica se usará la unidad activa en el momento de la instalación.

Un monitor de memoria sólo se echa en falta cuando se necesita, y esto sucede siempre que, por ejemplo, queremos manipular un programa que ya está corriendo en la memoria: modificándolo para probar, ajustar, buscar secuencias, cargar o salvar un bloque de código o de datos o de simplemente "cotillear", o bien tenemos un fichero en disco que precisamos analizar.

L - Solicita desinstalar el programa de la memoria.

QUE HACE

Al ejecutar RESIMEM se nos muestra una ventana dentro de la cual se observa un volcado en hexadecimal y en ASCII del contenido de la memoria (segmento:offset), pertenecientes al área que correspondería al siguiente programa a cargarse en la misma. La elección de esta posición inicial se debe a que parece la más util (algún valor inicial hay que darle).

El programa permite efectuar rápidos cambios directamente en memoria sin tener que recompilar

También se ve la dirección del segmento (mostrada en hexadecimal) y del offset (en hexadecimal y entre paréntesis en decimal) actuales. En la parte inferior de la pantalla se muestra un pequeño *help* resumido.

El programa nos permite modificar diréctamente el contenido de la memoria, tanto en hexadecimal como en ASCII, desplazarnos dentro del segmento, avanzar en bloques de 64 Kbytes y especificar nuevo segmento y offset.

Dispone además de cuatro opciones: rellenar la memoria con un dato constante, salvar un bloque de memoria a fichero, cargar un fichero en la memoria y buscar secuencias de datos.

Esta última opción se ha diseñado de manera que permita buscar datos expresados como bytes como caracteres o como palabras, permitiendo la mezcla de datos en la definición.

Asimismo, una vez hecha la definición se puede repetir la búsqueda de la siguiente coincidencia usando la combinación de dos teclas.

También permite ver, cuando se está en modo residente, el estado de los registros de la CPU al tomar el control.

DESCRIPCION FUNCIONAL DE RESIMEM

En primer lugar se comprueban los parámetros pasados para decidir que acción tomar. Si no se utiliza *R*, el programa se ejecutará apareciendo en la pantalla el volcado de la memoria, disponiéndose de las opciones usuales, excepto la información del estado de los registros de la CPU. Esto permite correr una "copia" del programa en modo "no residente".

Después se comprueba si existe ya una copia instalada en memoria. Para ello se obtiene el vector en uso de int 9 y se compara el texto de *copyright*. Este método suele ser más que suficiente para determinar si el programa ya reside en memoria: tiene el inconveniente o la ventaja, según nos convenga, de que si se instala otro programa que modifique el vector y lo ejecutamos de nuevo con orden de que quede residente, éste lógicamente no detectará la copia residente en memoria (esto también afecta a la desinstalación).

Si fuese imprescindible asegurarse de que no exista más de una copia en memoria se puede solucionar fácilmente "enganchándose" a un servicio de interrupción al que se le añada una función específica de comprobación de instalación.

Los más avispados notarán que el programa no se engancha a int 28h: esto se ha hecho adrede ya que el objetivo del programa consiste precisamente en utilizar al mínimo los recursos del sistema para poder ser lo más independiente de él. En programas que precisen un nivel de acceso mayor sí puede interesar engacharse a dicha interrupción o incluso disponer de un buffer para preservar el área de DOS completa y trabajar en modo "multitarea" del DOS (función 5D06h - GET ADDRESS OF DOS SWAPPABLE DATA AREA).

A continuación se inicializan algunas variables clave y se asigna la unidad de disco a usar para preservar los datos en pantalla cuando ocupan más que el buffer reservado a tal fin.

El programa RESIM.C (junto con RESI.ASM) puede servir de esqueleto para otros desarrollos

El programa reserva una relativamente pequeña cantidad de memoria convencional para almacenar la imagen en pantalla (normalmente una pantalla de texto) pero cuando se trata de pantallas gráficas que exceden el tamaño del buffer se recurre a guardarlas temporalmente en disco. Si no se utiliza el parámetro D para especificar la unidad de disco a usar en estos casos se asigna la unidad en uso en el momento de la instalación.

El fichero creado no se borra una vez recuperada la pantalla. Esto permite también usar el programa como un rudimentario salvapantallas.

INSTALACION DEL TSR

En caso de usar la opción *R* se salta a la rutina de instalación. En ella se obtiene el puntero a las variables del DOS incluyendo el área de intercambio para operaciones en "multitarea". Lo que nos interesa en concreto es obtener el puntero a las variables *error_crítico* y a la famosa *in_dos* (contador de reentradas al DOS). Estas se usarán para evitar entrar al



programa residente cuando se están utilizando los servicios del DOS.

También se obtiene el vector de int 13h y se le asigna una rutina propia que activará un *flag* al entrar a los servicios de disco con objeto de evitar reentradas accidentales.

Una opción podría ser permitir siempre la entrada y no habilitar las opciones de manejo de archivos en caso de que el DOS o los servicios de disco estén en uso en ese momento.

Como método de entrar al *tsr* se utiliza la combinación de tres teclas: CONTROL, ALT y M. Para ello se reasigna el vector de int 9 a nuestra rutina (analizada más adelante).

La instalación finaliza mostrándose un mensaje de aviso para acceder al *tsr*. En ese momento se devuelve el control al DOS y se queda a la espera de las teclas de activación.

ENTRADA AL TSR

El programa queda a la espera de que se produzca una interrupción de teclado. Cuando ésta se produce se procede a comprobar:

- En primer lugar el flag de control en uso del tsr.
- El código de tecla, que deberá corresponder al código de la tecla "M", en cuyo caso se verifica que:
  - Se comprueba en el área de datos del BIOS si se estába pulsando CTRL + ALT. Este método es cómodo y sencillo pero implica que el programa que actualmente esté ejecutándose permita la actualización de dicho dato. Normalmente esta situación se cumple, aunque no sucede lo mismo por ejemplo en programas de juegos en los que se "apropian" del control del teclado.
  - No se están usando los servicios BIOS de disco (int 13h).

Es importante evitar reentradas accidentales a los servicios del DOS y de disco del BIOS

  - No se están usando los servicios del DOS (int 21h) ni en situación "crítica" del DOS (ya que no son reentrantes).

    En el caso de que estén en uso los servicios del disco, del DOS o en situación de error crítico, no se permitirá el acceso al *tsr*, sonando un pitido de aviso de "no disponible".
- Se comprueba el Modo de Video activo en ese momento. Si éste no está en la lista de los admitidos (se soportan los modos de texto y gráficos "clásicos"). Se producen 3 pitidos de aviso por error y no se permite la entrada al *tsr*.
- Si todo va bien, se comprueba si la pantalla se salva en memoria o en disco. Si es en disco, en primer lugar se verifica que exista capacidad suficiente (en caso contrario se produce el aviso sonoro y no se entra en el *tsr*). El archivo se crea siempre en el directorio raíz con nombre _SCR_.$$$. La pantalla salvada se repondrá en el momento de salir del *tsr* pero, si se trataba de una pantalla gráfica, se asume que los colores serán los estándar ya que no se preserva la paleta de colores.

La opción de búsqueda permite encontrar estructuras de datos relativamente complejas

Una vez cumplidos los requisitos se entra al menú. Este es idéntico al que se muestra en modo no residente.

MENU DEL PROGRAMA

Al entrar al menú se limpia la pantalla. Normalmente se fueza a modo 3 (texto color 80x25) por hardware usando la rutina de la BIOS, pero si ya se estaba en dicho modo, sólo se procede a limpiar la memoria correspondiente a una pantalla de 80x25 caracteres por software. Este método evita comprobaciones adicionales para determinar si en realidad la pantalla contenía más de 25 líneas de texto (p.ej.: 80x43, 80x50, etc...) y el tener que disponer de un buffer mayor.

A continuación se procede a realizar un volcado de la memoria en pantalla a partir del segmento:offset actual (que inicialmente se asignó al segmento de entorno del siguiente programa en memoria).

Para cambiar de segmento y offset específicamente se usa la tecla *Inicio* (Home) apareciendo una pequeña ventana que nos solicita el nuevo segmento y offset. Si no se introduce ningún número, pulsándose la tecla *Enter* diréctamente, se mantendrá el segmento o el offset en uso.

Se admiten dos formatos para introducir los números: en decimal y en hexadecimal. El modo de introducir el formato hexadecimal es algo peculiar: se teclea en primer lugar la letra h (o H) seguida de los dígitos hexadecimales. La decisión de adoptar este formato estriba en que es relativamente rápido de teclear y a que, para simplificar, este método no usa la notación de ningún lenguaje específico.

A su vez se puede cambiar el offset simplemente desplazando el cursor con las teclas de dirección, o con las teclas de *página arriba/abajo*.

También se puede cambiar de segmento, en bloques de 64 Kbytes, pulsando la tecla *Control* al mismo tiempo que la de *página arriba/abajo*.

El volcado de la memoria se realiza en hexadecimal y ASCII. En el formato ASCII se dispone de dos opciones: mostrar todos los caracteres con su carácter gráfico asignado (por defecto) o mostrar sólo los caracteres "imprimibles" (códigos del 32 al 127), viéndose los restantes como un punto. Esta opción es intercambiable cada vez que se pulsa *ALT* y la tecla "A".

El contenido de la memoria se altera diréctamente y dependiendo de en qué zona esté el cursor: si se encuentra en la zona hexadecimal bastará con teclear dos dígitos hexadecimales y si está en la zona ASCII tecleando el nuevo o nuevos caracteres a insertar. Para cambiar de una zona a otra bastará pulsar la tecla de tabulación.

Si se está en modo residente al pulsar *ALT* y "R" aparecerá una ventana en la que se muestra el estado de los registros. Esta opción quizás no sea especialmente usada pero sirve de muestra de cómo crear una ventana con información útil.

OPCIONES

Se dispone además de cuatro opciones. Para acceder a cada una de ellas se debe pulsar la tecla *Insert*, apareciendo una ventana que esperará a la pulsación de la letra inicial.

- Fill Memoria
- Save Fichero
- Load Fichero
- Buscar Datos

Hay que ser especialmente cuidadoso al cargar ficheros muy grandes

Si se ha usado la opción de buscar datos en memoria, para realizar automáticamente la búsqueda de la siguiente coincidencia bastará con pulsar *ALT* y "S".

FILL MEMORIA

Usada para rellenar la memoria con un byte o carácter constante. Se nos pedirá el número de bytes o caracteres a rellenar y el byte o carácter de relleno.

Como esta opción es bastante destructiva sólo se permite el rellenado de la memoria en los 65536-1 bytes del segmento actual como máximo.

El valor de relleno se puede expresar como un número decimal, hexadecimal o como un carácter entre comillas.

SAVE FICHERO

Permite salvar un bloque de memoria a un fichero a partir de la posición apuntada por el cursor.

Aparece una ventana que nos pide el nombre del fichero (puede especificarse la unidad y la trayectoria) y la longitud en bytes.

Cuando se utiliza en modo residente puede servir como un rudimentario salvapantallas.

Si previamente se especificó la carga de un fichero, se nos mostrará el nombre (sólo el nombre y la extensión) y el tamaño del fichero como referencia. El nombre es obligatorio introducirlo: si se pulsa *Enter* sin haber tecleado nada más se considerará abortada la operación.

No ocurre lo mismo con los bytes a salvar: si se pulsa *Enter* sin teclear nada más se asumirá que se debe usar el tamaño anteriormente cargado.

LOAD FICHERO

Permite cargar un fichero en la memoria a partir de la posición apuntada por el cursor. Se nos pedirá el nombre del fichero (pudiendo especificarse la unidad y la trayectoria) y a continuación se nos pedirá el número de bytes a cargar, mostrándose en pantalla la longitud total del archivo. Si se pulsa *Enter* se asumirá que se desea cargar la longitud total.

Es preciso darse cuenta de que esta opción ha de usarse con cuidado ya que el programa carga el fichero en la memoria sin comprobar sobre qué lo está cargando, siendo responsabilidad del usuario el decidir si dispone de memoria suficiente o no (sobre todo trabajando en modo residente).

Con las versiones actuales del DOS (y programas como QEMM) es posible disponer de 634 Kbytes de memoria libre que suele ser más que suficiente para cargar grandes programas o bloques de datos para analizarlos con comodidad.

Una extensión que se le podría añadir al programa es un offset dentro del archivo a partir del cual se cargará en la memoria y que sería relativamente útil para ficheros de tamaño superior a los 600 Kbytes.

BUSCAR DATOS

Esta opción nos permite buscar datos con una gran flexibilidad: podemos hacerlo como bytes o ca-

racteres, series de números, cadenas o una mezcla de todo. Por ejemplo:

13,10,'cadena "muestra" ejemplo', hd,ha

o también:

3,10,"cadena 'muestra' ejemplo", ha0d

Esto dos ejemplos muestran que para buscar cadenas encerradas entre comillas se debe usar como delimitadores inicial y final dos comillas simples si en el texto hay comillas dobles y viceversa.

A su vez se observa que se permite buscar números de tamaño de palabra (ha0d en el ejemplo). Esto trabaja de un modo un poco *especial:* sólo se puede usar para valores numéricos en los que la parte alta de la palabra no sea 0, de lo contrario se tratará como si se especificara un simple byte o caracter, por ejemplo:

h0020 es lo mismo que escribir 32 o un espacio entre comillas simples o dobles.

Hay que hacer notar que la búsqueda es sensible al "tamaño" de los caracteres: "Texto" no es lo mismo que "texto".

Este mecanismo, bien entendido, permite buscar cadenas de datos complejas.

RUTINAS UTILES

En este programa se usan una serie de rutinas menores de posible uso con otros programas o lenguajes en los que se precise un tamaño reducido,

La opción de salvar permite tomar bloques de código para un posterior análisis

mayor velocidad o simplemente mayor flexibilidad o independencia. Entre ellas están:

IMPRESION DE CARACTERES.

La rutina de impresión de caracteres aquí usada tiene sus equivalentes en las suministradas con las librerías de "C" y "Pascal" por poner un ejemplo, sin embargo tiene la ventaja de un mínimo de código y un cómodo control del color, posicionamiento del cursor, etc.

Su idea de funcionamiento está basada en el método de impresión de caracteres del microordenador SPECTRUM (pura nostalgia) y permite incluir entre el texto que se va a mostrar una serie de códigos de control (algo parecido al ANSI). Estos permiten:

- Definir el color de la tinta y del papel (el fondo) tanto conjuntamente como por separado.
- Intercambiar el color de la tinta con el del papel.
- Definir la posición del cursor.
- Poner el cursor, dentro de la línea actual, en la columna especificada.

BEEPER Y RETARDO INDEPENDIENTES DE LA VELOCIDAD DEL PROCESADOR.

Esta rutina no tiene nada de misterioso; simplemente usa una rutina de retardo para obtener un sonido de duración ajustable pero independiente de la velocidad del procesador.

La rutina de retardo usa el contador del timer, leyendo la zona de memoria de la BIOS, para no depender de la velocidad del procesador.

Este método es sencillo y no requiere tener que "engancharse" a la interrupción 8 o a la 1Ch. Naturalmente, todo programa que tenga que engancharse necesariamente a alguna de estas dos interrupciones dispondrá de un mejor control.

IMPRESION DE BLOQUES GRAFICOS QUE USAN PLANOS DE COLOR.

La rutina *recu_pant* muestra el uso de volcado gráfico directo en pantalla para modos gráficos que usan 4 planos de color (16 colores en formato IRGB en EGA o VGA).

INPUT.

Permite una entrada de caracteres por teclado "controlada".

Estas y otras rutinas (que serán de mayor o menor interés, según los casos, que las que aquí se comentan) probablemente sirvan como una idea de partida para desarrollos más complejos.

COMENTARIO A RESIM.C

El programa muestra una serie de pequeños añadidos en el interface entre C y ensamblador: éstos (y probablemente algunos más, dependiendo del uso de los recursos del sistema) son necesarios ya que normalmente las rutinas de librería de C usan intensivamente el sistema en operaciones de entrada/salida.

Esto deberá tenerse en cuenta siempre que se acceda a las rutinas C en modo residente. Así, por ejemplo, la rutina que accede al *beeper* desde RESI.ASM, prepara un acceso más sencillo que la que accede a *resimem* debido a su acceso limitado. De hecho, no precisa salvar todos los registros, se ha hecho frente a posibles añadidos posteriores.

UNA NOTA FINAL

Para los que deseen incluir la gestión XMS un aviso: cuidado con las reentradas accidentales a los servicios XMS, ya que tienen el mismo problema que el DOS (al menos hasta la versión actual 6.2 del DOS) pues, como éste, utilizan un espacio de pila propio al que siempre se le asigna la misma posición inicial en cada solicitud. ■



MANEJANDO CLASES

Ignacio Cea

Por fin se ha introducido algo en el mundo de la programación bajo C++. Programar bajo ese lenguaje no supone conocer un conjunto de reglas y palabras clave sino también el empleo de ciertas maneras y formas que no pueden sino ser obtenidas a través de la experiencia. Esas maneras y formas vienen en llamarse: "Estilo C++".

El capítulo anterior no pretendía enseñarle esas maneras, sino adentrarle en el complejo mundo de la clase, algo que ni mucho menos, es cosa sencilla de explicar. En esta nueva parte vamos a ampliar los conceptos de la anterior entrega intentando que vea que se quiere decir cuando se habla de "Estilo C++".

ORDENANDO EL CODIGO

Antes de proseguir es necesario saber que en C++ es normal (no obligatorio) la existencia de 3 tipos de ficheros: ficheros *header,* ficheros *body* y ficheros *main.*

Los ficheros *header* contienen la definición de la estructura de una y sólo una clase. Los ficheros *body* contienen la definición del cuerpo de los métodos de una clase y, por último, el fichero main alberga la función de entrada al programa, *main.* Para construir una aplicación basta, por tanto, abrir un proyecto y colocar los ficheros *body* de las clases que intervienen en el mismo y el fichero *main* de entrada a la aplicación.

Esta separación no se hace por gusto, sino para que las clases puedan ser reutilizadas en otras aplicaciones con mucha mayor sencillez y rapidez. Todos y cada uno de los ejemplos de esta entrega han sido desarrollados mediante esa técnica.

Los ficheros *header* en C++ suelen terminar en .hh, en lugar del .h clásico precisamente para distinguirse de los del C; mientras que los ficheros body y main concluyen en la extensión .cpp por la misma razón.

Se va a entrar en el detalle de todos y cada uno de los ejemplos del capítulo, explicando las novedades del lenguaje que aportan.

En esta nueva entrega se van a introducir muchos nuevos conceptos, además de tratar de ahondar en todos aquellos otros que fueron expuestos en el anterior capítulo. Todo ello, siempre, a través de múltiples ejemplos y de una manera clara y fluida.

¿DONDE ESTAS QUE NO TE VEO?

El ejemplo que a lo largo de esta entrega va a ser expuesto, trata de mostrar en la pantalla un montón de objetos Rectángulo moviéndose a su libre albedrío.

Cada objeto rectángulo está caracterizado por una posición (que cambia conforme éste se va desplazando), un tamaño, un color y una dirección inicial de mo-

vimiento (que variará a medida que el objeto vaya chocando contra los límites del monitor).

Esas características son declaradas dentro de la clase (fichero Rectang.hh) tras la palabra clave *private.* Un miembro es *private* (en C++ las variables y métodos de una clase son miembros de ésta) cuando no se puede acceder a él desde otro sitio que no sea esa misma clase. Así, por ejemplo, el método *dibuja(),* también de la clase Rectángulo, podría cambiar, si lo quisiera, el valor de la variable color porque tanto uno como otro son miembros de la clase Rectángulo (están definidos dentro de ella). Sin embargo, en el *main* no se podría escribir una orden como rectangulo1.color (siendo rectangulo1 un objeto de la clase Rectángulo) pues se estaría tratando de invocar una variable privada desde fuera de la clase. Por otro lado si que podría escribirse algo como *rectangulo1.dibuja(),* ya que este miembro está declarado como público y, por tanto, es accesible desde el exterior de la clase.

En C++, el programador puede definir tres tipos distintos de ficheros: *Header, body y main*

Private y *public* son lo que en el C++ se denominan identificadores de acceso.

LA LEY DEL MINIMO ESFUERZO

Imagine una aplicación en la que se manejen simultáneamente 100 objetos Rectángulo y que el 90% de ellos sean blancos (color 15) y se muevan, inicialmente, hacia abajo y hacia la derecha (dx = 1 y dy = 1). Los tres últimos parámetros del constructor de cada uno de esos 90 objetos son exactamente los mismos (15,1 y 1). ¿Porqué no declararlos por defecto y evitar ponerlos en 90 de las 100 ocasiones que se cree un rectángulo?.

El C++ permite la declaración de los parámetros de un método con valores por defecto. Esto es, si al llamarlos no fuera especificado alguno de ellos, el compilador les asignaría los valores definidos en el prototipo como por defecto. En C++ una vez definido un parámetro por defecto, han de serlo, también, todos los que a éste sigan.

En el caso del Rectángulo se han declarado por defecto los tres últimos parámetros (color, dirección X e Y). Observe el fichero Rectangulo.hh para comprender mejor como se declaran.

MAS Y MAS RAPIDO

Si lee más detenidamente este fichero caerá en la cuenta de que hay un método cuyo prototipo está precedido de la palabra clave *inline* (el método *dibuja()*).

Inline solicita al compilador que cuando desde cualquier parte de la aplicación se llame a ese método no lo sustituya por una instrucción máquina *call,* sino que coja su código y lo coloque tal cual en el ejecutable. Estos métodos son, a grandes rasgos, algo así como una macro del preprocesador C (#define).

Los métodos *inline* son más rápidos que los tradicionales pero, si tienen demasiadas instrucciones, pueden llegar a alargar los ejecutables de manera sustancial. Por esta razón sólo se suelen emplear cuando el método no contiene más de un par de instrucciones. Además, los métodos *inline,* deben reunir ciertas condiciones como: no ser recursivos, no tener estructuras de bucle, no tener gotos, etc...

Recuerde que inline es un solicitud al compilador y no una orden. Sólo será atendido su requerimiento cuando las condiciones de memoria existentes así lo permitieran.

CREANDO RECTANGULOS

El constructor de una clase es un método, definido dentro de ella, que se ejecuta cada vez que un objeto de ese tipo es inicializado. Es un método parametrizable, ya que no todos los objetos se crean igual. Observe sino el caso de la clase ejemplo Rectángulo: cada objeto, aún siendo miembro de la misma clase, tiene un color una posición y un tamaño inicial diferente del resto de los de su mismo grupo.

Establecer un valor por defecto amplía la versatilidad de un método creado por el usuario

Esos parámetros se colocan, siempre, entre paréntesis tras el nombre del objeto creado: Rectángulo(1,1,5,5,2,0,-1). Observe la línea 17 del fichero Ejemplo1 para observar esta instrucción.

Los objetos también pueden ser creados dinámicamente. Para ello, como para cualquier otro tipo de variable, es preciso reservar memoria.

MALLOC, NEW

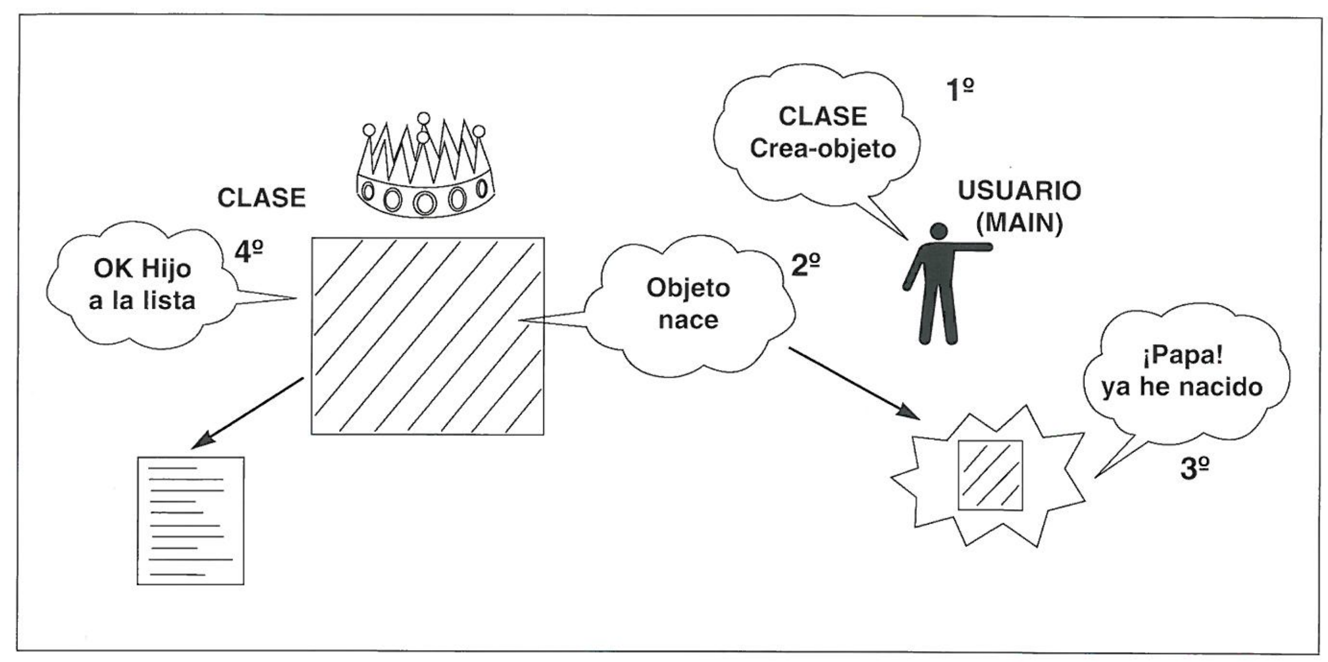
En C++ no se emplean jamás las ya vetustas funciones de reserva de memoria derivadas de *malloc* (farma-

CUADRO 1

Ficheros que integran el proyecto ejemplo del artículo:

Rectang.cpp

Ejemplo1.cpp



Cuadro 2: ¿Cómo se crea un objeto Rectángulo?.

lloc, remalloc, etc...). En su lugar se ha creado un operador que responde al nombre de *new*.

New reserva memoria para el tipo de dato que le sigue, sin paréntesis, y devuelve un puntero a la posición de memoria del hueco encontrado. Además, el tipo del puntero que *new* devuelve cuadra con el tipo solicitado por lo que no son necesarios los molestos cast a los que *malloc* nos tenía acostumbrados. En caso de que no existiera memoria disponible, *new* retornaría NULL(0l).

Puede también localizar hueco simultáneamente, para varios elementos de un mismo tipo. Para ello basta con colocar, tras el tipo y entre corchetes, el número de elementos deseado.

Las funciones constructoras son las encargadas de inicializar los datos pertenecientes a una clase

Si lo que se crea dinámicamente es un objeto (y sólo uno), se puede colocar, tras el nombre del tipo del objeto (que no es ni más ni menos que la clase a la que pertenece) y entre paréntesis, los parámetros necesarios para ejecutar el constructor e inicializar, de esta manera, ese nuevo objeto.

En la línea 27 del fichero Ejemplo1.cpp se muestra un ejemplo de cómo se reserva memoria dinámicamente para dos objetos de la clase Rectángulo.

DELETE CONTRA FREE

El antónimo de *malloc, free,* también ha dejado de existir. En su lugar C++ ha creado el operador *delete.* Este elimina el espacio de memoria reservado mediante *new.* Lo único que le acompaña es pues un puntero que señala la zona de memoria que se desea liberar, y que debe coincidir con el que en su momento nos devolvió el operador new, si queremos evitar "cuelgues" del sistema.

Si la zona correspondía a un objeto, antes de liberar el espacio por él ocupado, se ejecuta el destructor de la clase a la que pertenece para que realizar sus últimas peticiones.

New y *delete* no son compatibles con *malloc* y *free.* Es decir, la memoria reservada con new no puede liberarse con *free* y viceversa.

Observe las dos últimas líneas del fichero Ejemplo1.cpp donde se emplea este operador para borrar del mapa a los dos rectángulos creados dinámicamente.

LA VIRULENTA TRIBU DE LOS RECTANGULOS

Seguramente del capítulo anterior extrajo la idea de que una clase es, simplemente, un molde para la creación de objetos. En la mayor parte de las ocasiones así es. Sin embargo, en algunas otras class puede servir para mucho más.

Suponga que lo que se quiere es mover todos los rectángulos creados para una aplicación cualquiera. ¿Cómo debería ser el proceso?:

Una manera sería llamando al método mover de cada uno de los rectángulos de la aplicación (*n* órdenes del tipo *rectanguloxxx.mover()*). Otra forma podría consistir en pedir a alguien nombrado jefe de todos los rectángulos que moviera a todos sus súbditos. El jefe de todos los rectángulos es la clase a la que pertenecen.

Sin embargo, si todos los métodos que se definen dentro de una estructura *class* son sólo ejecutables por los ob-

jetos de esa clase y no por la propia clase (*class*)...¿Cómo puedo implementar esta segunda opción?.

Si a la definición de cualquier miembro de una clase se le antepone la palabra clave *static* el miembro pasará a ser inmediatamente compartido por todos los objetos de esa clase.

Por tanto, podríamos, colocar una variable estática dentro de la clase que se llamará numero_elementos, que fuera incrementada cada vez que un objeto naciera y decrementada cada vez que muriera. Cualquier objeto

Delete elimina el espacio dinámico de memoria reservado mediante la función estándar _new_

vivo podría, en cualquier instante consultar esa variable y saber cuántos hermanos conviven con él en el oscuro y frío display.

Esa variable sería, como un controlador (jefe) de objetos rectángulo.

¿QUE MAS DEBE SABER UN JEFE?

No olvide que el objetivo es crear un jefe de objetos que sea capaz, de moverlos por pantalla. Para poder mover todos los rectángulos, ese jefe debe saber como poco quiénes y cuántos son. El saber cuántos lo indica la variable comentada en el punto anterior. Por otro lado, el saber quiénes son puede hacerse a través de un array de punteros a los objetos rectángulo. Esto está declarado en el fichero Rectangulo.hh (static int numero_elementos y static Rectángulo *[__MAXRECTANGULOS__]). Es decir, con esas dos variables cualquier objeto puede saber no sólo cuántos hermanos conviven con él, sino además, cuáles son sus nombres.

Cuando se crea un objeto de esta clase el jefe debe, además, sumar uno al número de rectángulos existentes, y poder incorporar su nombre a la lista para así moverlo posteriormente a petición del usuario.

Por otro lado, también debe ser capaz de enterarse cuando alguien muere, ya que debe darlo de baja en la lista y no cometer así el error de mover a un muerto.

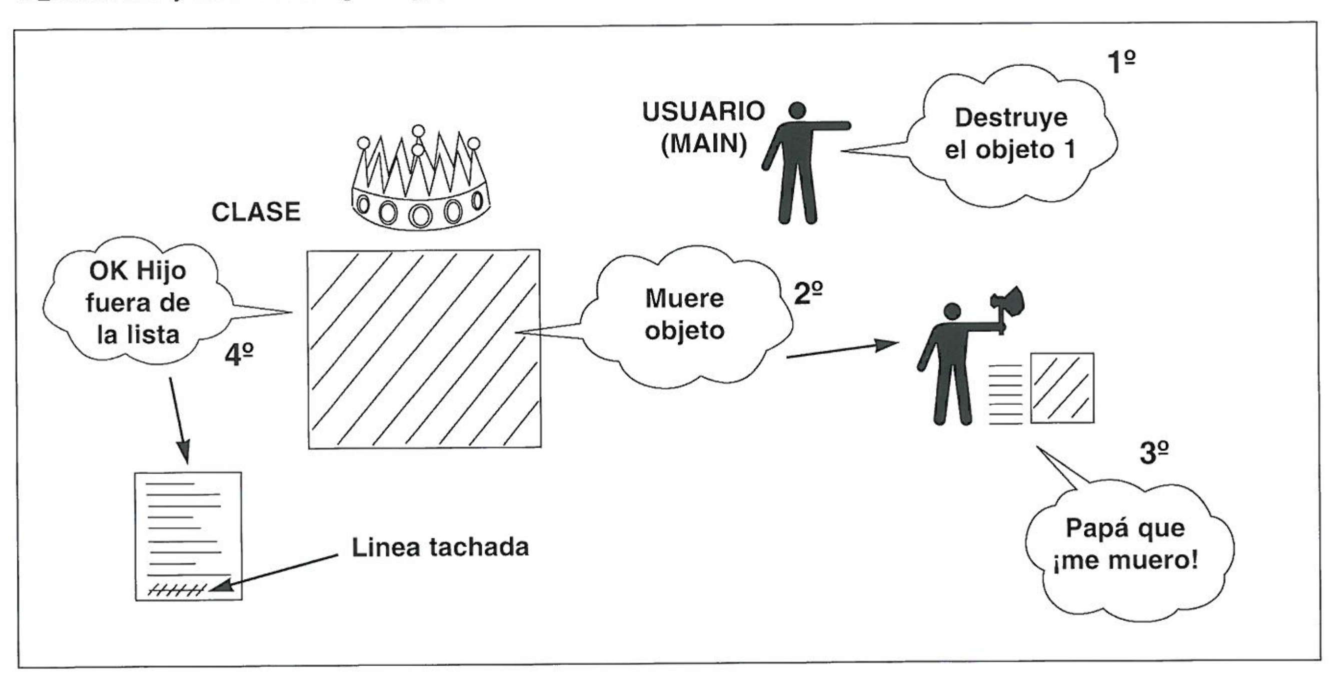
Por esta razón se implementan los métodos *inserta* y *elimina*, declarados junto a manipula_elementos al final del fichero Rectang.hh. Todos esos métodos son, por así decirlo, las formas que tiene de actuar el jefe (la clase) sobre sus objetos súbdito.

LA VIDA DE LOS OBJETOS

Pero... ¿Cómo se enterá la clase del nacimiento o muerte de un objeto? Pues se lo dice el mismo objeto. No olvide que cuando un objeto nace, se ejecuta siempre el constructor y que cuando muere lo hace el destructor. Luego dentro del constructor y destructor habrán de incorporarse llamadas a los métodos *inserta* y

Mediante la función destructora, el programador puede liberar memoria y concluir el trabajo de una clase

elimina respectivamente, que son como se ha dicho antes, la forma que la clase tiene de incorporar o eliminar miembros a su lista.



Cuadro 3: ¿Cómo se destruye un objeto Rectángulo?.

Para invocar a un método que no pertenece a la clase (y no a algún objeto) sólo tiene que colocar, delante del método, el nombre de la clase seguido de "::". Observe el formato de la llamada en las primeras líneas del fichero Rectang.cpp.

Pero... ¿Cómo le dice un objeto a su clase (jefe) que es él y no otro quien debe ser insertado (Rectangulo::inserta) en la lista de rectángulos a mover?.

Inline optimiza en velocidad las llamadas a los métodos creados por el usuario

Para estos casos se ha definido en C++ la palabra clave *this*. Es el nombre de un puntero definido por defecto dentro de cualquier objeto y que apunta a sí mismo...

Por tanto, desde el constructor de un objeto debería decirse algo así como: ¡Oye jefe, inserta en tu lista *this* (que soy yo) porque estoy naciendo! Y desde el destructor la conversación análoga sería: ¡Oye jefe, elimina de tu lista *this* (que soy yo) porque me muero!

Observe aquellas líneas del constructor y del destructor que ejecutan esas acciones.

PONIENDO EN MARCHA AL JEFE

Las variables propias de cada objeto pueden ser inicializadas desde dentro de su constructor, pero...¿Desde dónde son inicializadas las variables de una clase? (las que empiezan por *static*).

Observe la parte superior del fichero Rectang.cpp. Las variables de una clase se inicializan como si fueran nuevamente declaradas, pero dándole además un valor de origen.

Ya que un mismo nombre de variable puede aparecer dentro de muchas clases distintas, una ventaja de encapsular información, es necesario en esa inicialización anteponer al nombre de la variable el nombre de la clase a la que pertenece seguido de "::". Con esto evitaremos que el compilador se haga un lío.

DIBUJANDO RECTANGULOS

La forma de dibujar un rectángulo (método *dibuja()* de la clase Rectángulo) será, seguramente, muy familiar a aquellos lectores que hayan programado alguna vez a un nivel cercano a la máquina.

El dibujar un rectángulo se hace directamente sobre la memoria de vídeo, que en un ordenador a color ocupa la posición RAM 0xb8000. Si usted no dispone de tarjeta a color (que no de monitor a color) sustituya el MK_FP del método por MK_FP (0xb000,0x0000);.

LOS ARRAYS DE OBJETOS

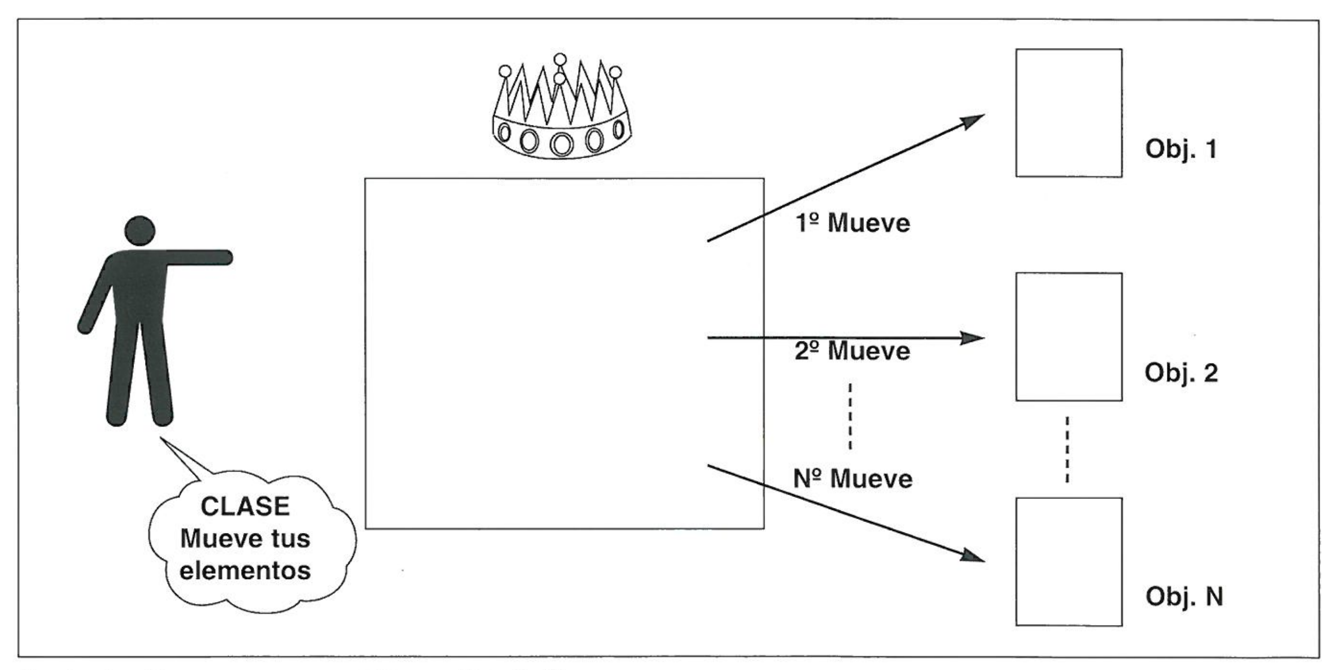
Igual que dentro de cualquier aplicación C o C++ se pueden crear arrays, por ejemplo, de número enteros o de caracteres, en un programa C++ se pueden crear arrays de objetos. Y al igual que aquellos, estos también pueden ser inicializados.

Inicializar un array de objetos supone inicializar cada uno de los objetos que lo integran. Para inicializar un objeto dentro de un array basta invocar directamente al constructor del mismo.

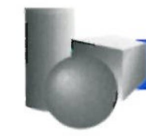
Observe las líneas 18 a 24 del fichero Ejemplo1.cpp para ver y comprender cómo se realiza esa inicialización.

SOBRECARGA

Seguramente ya habrá advertido que dentro de la clase Rectángulo hay un método sobrecargado: *dibuja*. Un rectángulo puede ser dibujado o bien en el color es-



Cuadro 4: ¿Como se mueven nuestros rectángulos?.



pecificado por sus variables internas o bien en otro cualquiera.

El método *dibuja,* con parámetro, se invoca siempre que se desea borrar el rectángulo de la pantalla en el método *mover* y en el destructor. Por este motivo siempre es llamado con el valor 0 como parámetro (el fondo se supone siempre negro). Por otro lado el otro método *dibuja,* sin parámetros, es llamado cuando se desea dibujar el rectángulo en el color elegido durante su construcción, señalado por la variable interna color.

El método dibuja se invoca siempre que se desea borrar el rectángulo de la pantalla en *mover* y en el destructor

Si esto es así, ¿por qué no se colocado ese parámetro con un valor por defecto de 0? ¿Acaso no era ese el motivo dado anteriormente para poner parámetros por defecto?

Si se hiciera eso al invocar el método *dibuja(),* el compilador no sabría si emplear el que está definido con un valor por defecto de 0 en su parámetro o el que no tienen ni tan siquiera definido uno. Pruebe a hacerlo y verá cómo se queja.

¿POR QUE ESE EXTERN TAN RARO?

Ya habrá caído en la cuenta de que siempre que se realiza un include sobre librerías estándar del C se antepone la palabra reservada *extern.*

Las funciones del C y del C++ no son compiladas de igual manera (en el C no hay ni sobrecarga, ni funciones dentro de clases, etc...). Si dentro de un programa C++ se compilaran las funciones de librería C al estilo C, el *linker* no sería nunca capaz de encontrarlas.

Por este motivo los programas C++ deben prevenir al compilador de que lo que ahora viene está definido al estilo C, de ahí la curiosa forma de esos *includes.*

La mayor parte de los compiladores consideran esta opción ya por defecto y no es necesario, por tanto, el decírselo explícitamente. Sin embrago, se ha preferido mantenerlo así por compatiblidad entre distintos compiladores.

UN CONSEJO

Puede ser, y no le culpo, que tras leer estas líneas tenga un profundo lío sobre el orden en el que dentro del C++ se ejecutan las cosas. Sólo Programadores le aconseja correr el ejemplo paso a paso (tecla F7 en Borland C++) y seguir la ejecución de la aplicación desde el principio del *main.*

¿COMO COMPILAR EL EJEMPLO?

El ejemplo de este artículo han sido desarrollado bajo el entorno Borland C++ 3.1. Sin embargo, la migración a otros entornos no debe suponer ningún quebradero de cabeza. Quizás lo único mencionable es que el fichero proyecto ejemplo de este artículo sólo puede ser abierto bajo ese entorno, por lo que si usted trabaja bajo otro sistema tendrá que crear el fichero proyecto a partir de la lista que se ofrece en el cuadro 1. Una vez abierto el fichero proyecto ejemplo correspondiente podrá correrlo de la forma habitual (CTRL-F9 en Borland C++).

LOS DEBERES

Bueno, todo lo expuesto está muy bien, pero no sirve de nada si usted no lo practica. Esta sección tiene por objetivo darle unas ideas de por dónde debe profundizar en sus investigaciones. Intente responder a las siguientes preguntas que, por supuesto, ¡no son nada fáciles!

1.- Nuestra clase Rectángulo (el jefe) sólo puede reconocer un máximo número de rectángulos. ¿Podría inventar algún esquema C++ para que en lugar de emplear un array como almacenamiento de direcciones se empleará una estructura dinámica como una lista, por ejemplo?

2.- ¿Qué sucede si no inicializó las variables de clase?

3.- Tal y como está definida la clase...¿Es obligatorio poner private? ¿Y el public?¿Por qué?

Para ejecutarse paso a paso un programa en Borland C++, debe pulsarse la tecla F7 en la función *main*

4.- ¿Qué hace la siguiente instrucción? : new char *[10];

5.- ¿Podrían, en esta aplicación, definirse los métodos de clase : *inserta* y *elimina* como privados? ¿Por qué?

6.- ¿Podría crear un objeto *Rectángulo* dinámicamente, sin invocar al constructor?, es decir, ¿podría escribir una orden como: Rectángulo *rect1 = new Rectángulo? ¿Por qué? ¿Qué necesitaría para poderlo hacer?

EPILOGO

En la próxima entrega trataremos en profundidad tanto el tema de la sobrecarga, como el apasionante mundo del "dinamic binding" o enlace dinámico que, junto a los "template" (que llegarán en dos meses) elevan al C++ a su máxima potencia. Pero eso es otra historia de momento. ■

DLLs SIN DOLOR

César Astudillo

Las bibliotecas de enlace dinámico (DLLs) constituyen una pieza fundamental del sistema operativo OS/2. Después de conocer sus características, posiblemente decida que gran parte del código de su próxima aplicación debe residir en una DLL.

Hace no tanto tiempo, cuando el MS-DOS sin Ms-Windows era prácticamente el único entorno para PC en el que se podía escribir una aplicación y conseguir que alguien la comprara, ya existía la necesidad de utilizar servicios comunes e independientes de las aplicaciones: acceder a bajo nivel a los recursos de la red local, efectuar llamadas a un manejador de comunicaciones, o utilizar algún dispositivo de hardware proporcionado por terceros. Normalmente, este tipo de servicios se implementaba -y aún se sigue haciendo así- por medio de un programa del tipo TSR (Terminate and Stay Resident) que dejaba instalada una interrupción a la que se podía llamar desde las aplicaciones. Quien haya profundizado en estas prácticas conocerá los problemas que se producían: consumo de memoria, conflictos con las interrupciones... Para el desarrollador que ha migrado desde este entorno a OS/2, el desarrollo de aplicaciones avanzadas se ha simplificado extraordinariamente. El papel central de las bibliotecas de enlace dinámico en este sistema operativo es una más de las características que intervienenen en esta simplificación, al tiempo que mejoran cualitativamente el rendimiento general del sistema.

NO ESTAMOS SOLOS

Por su parte, Ms-Windows guarda de su origen común con OS/2 grandes coincidencias en el concepto de DLL y sus aplicaciones, por lo que mucho de lo que se va a contar aquí es válido también para dicho entorno. Así, en las páginas que siguen se explicará el sentido que tienen las DLLs en el escenario del desarrollo de software; un terreno en que la coincidencia entre los sistemas operativos es prácticamente completa, y la implementación particular de una DLL y de los programas que hacen uso de ella siguen una misma filosofía.

CONOCER LO VIEJO PARA ENTENDER LO NUEVO

Para comprender el funcionamiento de una biblioteca de enlace dinámico, puede ser de ayuda el estudio de un concepto complementario, anterior a él, y más conocido: las bibliotecas de enlace estático (lo que muchos conocen sencillamente como “librerías”). Posiblemente sepa que una biblioteca de enlace estático es un fichero de extensión .LIB en el que se condensan porciones de código objeto que pueden pasar a formar parte de un pro-

grama. Cuando el desarrollador quiere incorporar alguna de las funciones contenidas en una biblioteca en su aplicación utiliza el enlazador -LINK-. Este detecta entonces en el código usuario las referencias a las funciones o elementos de datos de la biblioteca, y las sustituye por las direcciones que se les han asignado en el código ejecutable. A este proceso se le denomina *resolver las referencias*.

A este tipo de enlace se le llama enlace estático porque el proceso de resolución tiene lugar en tiempo de montaje. Es LINK quien se encarga de ello en la etapa final de creación del fichero ejecutable. Una vez montado, el código procedente de las bibliotecas pasa a formar parte de él, y es indistinguible del resto. El programa resultante es autosuficiente: contiene todo lo necesario para trabajar.

Este método de enlace tiene sus ventajas: es sencillo y no hace que el programa final sea dependiente de otros ficheros que se deban instalar. No en vano ha sido durante mucho tiempo la única manera de combinar distintas piezas de código. Sin embargo, también presenta varios inconvenientes que no lo convierten en el más aconsejable en ciertos casos.

DLLs: EL 'SPRAY ANTIESTATICO'

Por una parte, existe el problema del espacio. Si hay dos programas en el sistema que utilizan la misma biblioteca, el código de dicha biblioteca está presente en el fichero ejecutable de ambos programas, aunque el código sea exactamente el mismo. Esta redundancia significa más gasto de espacio en disco cuando los programas se distribuyen e instalan, y mayor gasto de memoria si ambos se ejecutan simultáneamente en el sistema. Y no olvidemos que, siendo OS/2 un sistema multitarea, esta última situación se puede dar con frecuencia.

También hay una dificultad adicional que afecta a la distribución y mantenimiento de los programas. Cualquier cambio en el código de la biblioteca -la corrección de un error o un cambio de versión- haría necesario volver a montarlo completo mediante LINK. Si los programas que hacen uso de la biblioteca son varios, la operación de cambio de versión se vería aún más enmarañada.

Enunciado el proceso de enlace estático, es el momento de tratar el método que constituye su alternativa: el enlace dinámico.

DINAMISMO AL PODER

Una biblioteca de enlace dinámico es un fichero con extensión DLL (Dynamic Link Library). En él se encuentran porciones de código disponibles para su uso desde otros programas, al igual que ocurre con las bibliotecas de enlace estático. La diferencia con estas últimas reside en el momento en que se efectúa el proceso de resolución expuesto unas líneas más arriba. En el enlace dinámico este proceso se deja para el momento en que se ejecuta el programa.

De este modo, la DLL está instalada en algún lugar del disco, y el programa o programas que hacen uso de ella pueden estar en cualquier otra parte. El programa contiene información de enlace que especifica en qué DLL se encuentra el código preciso, y cuando pasa a ejecución, solicita al sistema operativo la carga de dicha DLL. El S.O. busca la DLL en los directorios que se especifican en la sentencia LIBPATH del fichero CONFIG.SYS, y si la encuentra, la carga en memoria. Seguidamente, resuelve ya en memoria las referencias a la DLL contenidas en el código del programa usuario, especificando las direcciones correspondientes a donde la DLL está cargada. A partir de este momento, el programa puede utilizar el código de la DLL.

BUENAS, BONITAS Y BARATAS

Si un segundo programa -o una segunda instancia del mismo programa- entra en ejecución simultáneamente con el primero, OS/2 detecta que la DLL ya está en memoria, y no carga una segunda copia para uso del segundo programa, sino que utiliza la que se cargó para uso del primero. De este modo se hace un uso más racional de los recursos.

Por otra parte, cualquier cambio en el código de la DLL hace necesario simplemente cambiar el fichero DLL correspondiente. No es necesario hacer ningún cambio en el código de los programas usuarios en tanto no cambien los parámetros de entrada de las funciones. Desde el momento en que se cambie la DLL, los nuevos programas, al ejecutarse, se comportarán de manera distinta, reflejando los cambios.

La utilidad de este sistema se hace evidente cuando en un proyecto de desarrollo se debe contar con software aportado por terceros. Las DLLs y los programas que hacen uso de ellas se pueden instalar por separado y pueden tener políticas de distribución y actualización diferentes. Si, por ejemplo, su última aplicación de gestión para supermercados utiliza una DLL de manejo de lectores de códigos de barras aportada por un tercero, y meses después de su implantación, el cliente decide utilizar un nuevo modelo de lector, sólo necesita instalar la nueva DLL correspondiente sustituyendo a la antigua. Con ello habrá logrado la adaptación a un nuevo dispositivo sin tocar ni un byte de su aplicación.

DLLs HASTA EN LA SOPA

Aunque no lo parezca, OS/2 utiliza DLLs para casi todo. Le sorprenderá saber que incluso un sencillo programa del tipo "¡Hola, mundo!" en OS/2 hace uso de DLLs para su funcionamiento. Esto es así porque funciones tan básicas como *printf()* o *fopen()* hacen uso a su vez de las funciones del API de OS/2, y dichas funciones están implementadas en forma de DLL; en este caso, en el fichero DOSCALL.DLL. El comando *pstat* de la línea de comandos del OS/2 puede mostrar las DLLs de las que está haciendo uso un programa durante su ejecución. Un sencillo programa de Presentation Manager puede

estar haciendo referencia a muchas DLLs distintas relacionadas con el uso de la pantalla, paso de mensajes, gestión de ventanas... Examine su último programa con *pstat:* ¿A que no sabía que usara tantas DLLs?

EL QUE PARTE Y REPARTE...

Cuando una misma DLL es utilizada concurrentemente por varios programas, o por varias instancias de un mismo programa, el desarrollador tiene un alto grado de control sobre lo que ocurre -la forma en que se organizan y se reparten en memoria los datos de trabajo de la DLL-. Estos pueden replicarse automáticamente para cada uno de los procesos que están haciendo uso de la DLL, en cuyo caso, los cambios que se hagan como consecuencia de su uso desde cada proceso quedarán circunscritos al ámbito de éste. También puede existir una sola instancia de dichos datos en memoria aunque sean varios los procesos que referencian la DLL, con lo que serían compartidos entre los distintos procesos. Incluso se pueden repartir los datos de trabajo de la DLL en varios segmentos con características distintas, dependiendo de si se desea que el dato se comparta entre los procesos o sea privado para cada uno de ellos.

TIEMPO DE CARGA VERSUS TIEMPO DE EJECUCION

OS/2 también permite elegir entre realizar el proceso de enlace con la DLL en tiempo de carga, o en tiempo de ejecución. En el primer caso, la DLL se carga en memoria al mismo tiempo que lo hace el programa usuario, y es en ese momento cuando se resuelven las referencias. Asimismo, la DLL no se retira de la memoria hasta que el programa usuario termina. En el caso del enlace en tiempo de ejecución, el programa cargará la DLL en el momento que designe el desarrollador -cuando esté especificado así en el programa-, no necesariamente al inicio del mismo. Lo mismo ocurre con el momento en que se libera o descarga la DLL. En principio, un programa que haga uso de una DLL mediante un método de enlace en tiempo de ejecución puede hacer uso de varias DLLs por turno, teniendo cargadas en cada momento sólo las que esté utilizando. Incluso el nombre de la DLL a utilizar puede decidirse en tiempo de ejecución.

OS/2 permite elegir entre realizar el proceso de enlace con la DLL en tiempo de carga o de ejecución

Así, volviendo al ejemplo de la aplicación de supermercados, puede ser posible tener varias copias de la misma aplicación funcionando en distintos puestos de trabajo y utilizando modelos distintos de lector de código de barras, correspondiendo a cada modelo un nombre de DLL distinto. El modelo de lector que se utiliza en cada puesto podría especificarse y modificarse dinámicamente por medio de un menú de configuración.

ABRIENDO Y CERRANDO

Es muy común que la utilidad de una DLL sea facilitar a los programas usuarios el uso de un recurso, como podría ser un dispositivo o un servicio. En ese caso, antes de dar servicio al primero de dichos programas usuarios, la DLL debería reservarse dicho recurso, y análogamente, en el momento en que la DLL se vaya a retirar de la memoria porque se quede sin usuarios, debería liberar el recurso reservado. Estas tareas podrían ser realizadas por los programas usuarios a través de funciones especiales de la DLL que dichos programas usuarios deben llamar explícitamente en el momento oportuno, o bien se pueden localizar en puntos reservados como funciones de inicialización y terminación, a donde el propio sistema operativo realizaría la llamada automáticamente. OS/2 permite especificar estas funciones de inicialización y terminación.

POR FIN LA PRACTICA...

El ejemplo práctico que acompaña a estas líneas pone en juego varios de los conceptos que se han esbozado hasta ahora. Se trata de crear una DLL para un servicio de configuración dinámica de los mensajes de los programas, llamada MENSAJE.DLL.

La DLL MENSAJE permite agrupar en un solo fichero de texto, fácilmente modificable, las constantes del tipo cadena que se utilizan en un cierto número de programas. Este fichero se localizará en el directorio desde el que se ejecuta el programa, y todos los mensajes que utiliza el programa se referencian por medio de claves alfanuméricas o mnemónicos. MENSAJE.DLL se encarga de buscar el mnemónico elegido en el fichero de mensajes y devolver el mensaje correspondiente. De este modo, al tener todos los mensajes del programa separados en un fichero de texto, resulta mucho más fácil modificar los textos o traducirlos a otro idioma que si estuvieran embebidos en el programa (¡No olvide que el mercado nacional incluye el castellano, el euskera, el catalán y el gallego!). Además, pueden ser varios programas distintos o varias instancias distintas de un mismo programa en ejecución los que hagan uso de un mismo fichero de mensajes. Los mensajes sólo estarán cargados en memoria una vez, con independencia del número de programas que hagan uso de la DLL.

EL QUE AVISA NO ES TRAIDOR

Se ha escogido esta aplicación para la DLL de ejemplo porque se trata de una idea sencilla para cuya implementación es necesario desarrollar muchas de las ideas de las que se ha hablado más arriba. La intención es ante todo didáctica. Por tanto, cuestiones como el modo de cargar el fichero en memoria y de buscar las claves en el momento que se solicita un mensaje (detalles independientes del tema de este artículo) se han re-

suelto de manera que el código sea lo más corto posible: los mensajes se cargan en una matriz de tamaño predeterminado, y la búsqueda es secuencial. Por tanto, esta porción de código en particular está implementada de modo ineficiente, y no recomendamos su uso para otro fin que no sea el de aprender a construir bibliotecas de enlace dinámico, que es para lo que se ha creado.

El fichero MENSAJE.C contiene el código principal de la DLL. En MAKEFILE se puede apreciar como al compilar y montarlo con las especificaciones contenidas en el fichero de definición MENSAJE.DEF, se genera el fichero MENSAJE.DLL. Este se deberá copiar a cualquier directorio comprendido en la sentencia LIBPATH del CONFIG.SYS. Existen dos programas de ejemplo que hacen uso de MENSAJE.DLL: PROG1.EXE y PROG2.EXE. El primero utiliza la DLL mediante el procedimiento de enlace en tiempo de carga, y el segundo, mediante enlace en tiempo de ejecución.

Observará que MENSAJE.C no contiene función *main()*. Al igual que ocurre con casi todas las bibliotecas de enlace estático, se espera que la definición de esta función corra de cuenta del programa usuario. En cambio, en una DLL sí que existen una o varias funciones o elementos de datos que, al igual que *main()*, tienen una consideración distinta al resto: se trata de los puntos de exportación. Estos serán los puntos de entrada por los que se presta acceso a la DLL desde el exterior: las "ventanas" a través de las que los programas usuarios podrán utilizar la DLL. Los puntos de exportación pueden ser funciones y elementos de datos; en este último caso deberán ser datos estáticos, por lo que no se puede exportar variables automáticas. De cualquier manera, exportar elementos de datos no es aconsejable desde el punto de vista metodológico: si se desea que un programa usuario lea o escriba en un elemento de datos residente en la DLL, es más recomendable que se escriba una función de exportación que realice esta lectura o modificación. Esto ayudará a encap-

Los puntos de exportación de la DLL pueden ser funciones y elementos de datos

sular mejor el código de la DLL y la hará menos sensible a efectos colaterales debidos a un uso indebido de los elementos de datos por parte del programa usuario.

CUIDANDO LA BALANZA DE EXPORTACION

Cuando diseñe los parámetros de entrada y salida de las funciones de exportación de una DLL, piense que está definiendo el interface de usuario de dicha DLL, la forma en que ésta se relacionará con los programas que hacen uso de ella. Procure que haya el menor número posible de funciones y que el interface sea flexible: como se ha visto, si cambia la implementación interna de la DLL, no es necesario que cambie los programas usuarios, pero si se ve obligado a cambiar el interface externo no tendrá más remedio que cambiar los programas usuarios de modo que se adapten a dicho cambio, con lo que estaría perdiendo una de las ventajas principales del uso de DLLs. Por tanto, piense en crear el menor número posible de interdependencias entre el programa usuario y la DLL, y diseñe el interface con mentalidad de futuro. En metodología orientada a objetos, esta reco-

Se debe procurar que las funciones de exportación de la DLL sean las menos posibles

mendación se podría expresar en términos de minimizar el acoplamiento entre objetos. Recomendación que se ha seguido casi con exceso de celo en MENSAJE.DLL, puesto que sólo tiene una función de exportación: la función *Mensaje()*, que toma como único parámetro el mnemónico del mensaje y devuelve un puntero a la cadena que contiene el texto de dicho mensaje.

¿Cómo se distinguen las funciones de exportación del resto de las funciones? La técnica básica es incluir sus nombres en el apartado EXPORTS del fichero de definición. Se puede comprobar como la función *Mensaje()* se ha añadido a dicho apartado en el fichero MENSAJES.DEF.

La DLL MENSAJE debe leer la lista de mnemónicos y sus mensajes asociados del fichero MSG.TXT que se encuentre en el directorio actual. Cada línea de MSG.TXT tiene el formato:

```
mnemónico=mensaje<retorno de carro>
```

Se observará que el código de la función *Mensaje()* no puede ser más sencillo: recorre la matriz de estructuras mensajes donde cada una de ellas -del tipo definido mensaje- contiene un mnemónico y su mensaje asociado, hasta que encuentra el mnemónico buscado. En ese momento devuelve el mensaje correspondiente. Si no lo encuentra devuelve el último mensaje del fichero, por lo que al escribir el fichero se ha de prever esta circunstancia y reservar la última línea para el mensaje correspondiente al concepto "mensaje desconocido".

EL INICIALIZADOR FANTASMA ATACA DE NUEVO

Pero ¿cómo han ido a parar los datos del fichero MSG.TXT a la matriz mensajes? El programa usuario no se encarga de ello explícitamente, sólo de hacer uso del servicio. Por el contrario, el propio sistema operativo se encarga de llamar a la función que lo hace. También se ocupa de liberar la matriz cuando la DLL se descarga de memoria. Esto se consigue mediante una función lla-

mada *_DLL_InitTerm()*. Dicha función existe por defecto en toda DLL, aunque el desarrollador puede escribir la suya propia, como se ha hecho en MENSAJE.DLL. Para que no exista conflicto entre la *_DLL_InitTerm()* escrita por el usuario y la que el sistema asigna por defecto, es necesario añadir /NOE a las opciones de montado, como se aprecia en MAKEFILE. Esta opción da prioridad a las funciones definidas en el código central sobre las que se encuentren en las bibliotecas estándar.

La función *_DLL_InitTerm()* es llamada por el sistema operativo cuando la DLL se inicializa -se carga por primera vez- y cuando se termina -se descarga de la memoria-. En el fichero DEF asociado a una DLL se puede especificar si se llamará también cuando entre un nuevo programa a hacer uso de la DLL y cuando un programa deje de utilizarla. En la sentencia LIBRARY se especificará INITGLOBAL en el primer caso, y INITINSTANCE en el segundo. Esto último es lo que se ha hecho en MENSAJE.DEF. El parámetro *ulTerminating*

Existen llamadas para descargar temporal o permanentemente la DLL de la memoria

vendrá con el valor TRUE en caso de que el sistema operativo indique la terminación de la DLL, y con el valor FALSE en caso de que se trate de la inicialización. El valor de retorno que *_DLL_InitTerm()* debe proporcionar será TRUE en caso de que la operación haya tenido éxito, y FALSE en caso contrario.

Se ha utilizado un pequeño truco en la función *_DLL_InitTerm()* de MENSAJE.DLL: en la zona de datos comunes a todos los procesos usuarios se ha colocado un contador de usuarios llamado *usuario*. Lo primero que se hace en la inicialización es incrementar el contador, y en la terminación se decrementa. De este modo, la DLL sabe si el usuario es el primero en entrar, y si es el último en salir. Si es así, además de llamar a las funciones de inicialización/terminación de instancia, llama a las globales. Así, MENSAJE.DLL tiene dos funciones de inicialización y dos de terminación. Las funciones *InicializacionGlobal()* y *TerminacionGlobal()* se llaman sólo al entrar el primer usuario y al salir el último: las funciones *InicializacionInstancia()* y *TerminacionInstancia()* se llaman siempre que entra o sale un nuevo usuario. MENSAJE.DLL sólo necesita funciones de inicialización y terminación global, pero se han incluido también las de instancia para proporcionar un esqueleto válido para cualquier DLL más avanzada.

Obsérvese que en el código de inicialización y terminación se llama respectivamente a las funciones *_CRT_init()* y *_CRT_term()*. Estas funciones se encuentran en el código de inicialización/terminación por defecto, y se encargan de inicializar y terminar los servicios asociados a las bibliotecas estándar de C: posibilitan el funcionamiento de funciones como *printf()* o *scanf()*, por ejemplo. Como MENSAJE.DLL sustituye al código de inicialización/terminación por defecto, debe suplirla en esta tarea.

ENSEÑANDO A COMPARTIR

Se ha mencionado antes que existe una zona de datos comunes. Para explicar esto conviene recordar lo que se ha dicho más arriba sobre la flexibilidad que permite OS/2 en cuanto a la definición de las características de los objetos de datos en una DLL. En el caso de MENSAJE.DLL, y como se puede comprobar en MENSAJE.DEF, se ha utilizado un criterio mixto, que es el que proporciona mayor potencia. Como indica la cláusla DATA MULTIPLE NONSHARED, los datos estáticos que se utilicen en la aplicación se replicarán, por omisión, para cada uno de los procesos usuarios, con lo que su contenido será privado para cada uno de ellos. Esto permitirá evitar conflictos por interacciones indeseadas entre procesos. Si se desea que los datos sean compartidos por defecto, la cláusula correspondiente sería DATA SINGLE SHARED, aunque en ese caso habría que cuidar que todo el código de la DLL sea reentrante para evitar las interacciones citadas.

Como todos los datos estáticos son individuales por omisión, se ha definido un segmento de datos especial llamado COMUN, donde residen los datos que se desea explícitamente que se compartan. Esta es la zona de datos comunes que se citaba más arriba. Se define en la cláusula COMUN CLASS 'DATA' SHARED del apartado SEGMENTS en MENSAJE.DEF. Con ello se establece que dicho segmento ha de ser compartido entre los distintos procesos que hagan uso de la DLL. Por su parte, en el fichero MENSAJE.C se ha reservado una zona para declarar los objetos que se desea figuren en dicho segmento. Para ello se encierra entre dos directivas *#pragma data_seg()*: la primera especifica el segmento COMUN, disponiendo de este modo que todos los objetos de datos estáticos que se definan después de la directiva pertenezcan a dicho segmento, y la segunda, sin parámetros, indica que se vuelva al segmento de datos por omisión.

En MENSAJE.C se han incluido algunas llamadas a *printf()* que ayudan a saber en qué estado de ejecución está la DLL en cada momento. Todas ellas están encerradas en directivas de compilación condicional *#ifdef*, de modo que basta con suprimir la opción /DTRACK_INITTERM de las opciones de compilación para que dichos mensajes no aparezcan.

UTILIZANDO UNA DLL

Hasta ahora se ha definido de forma muy somera los elementos principales de la creación de una DLL. Ahora es el momento de crear un programa que la utilice.

LA SENCILLEZ DE IMPLIB

La forma más sencilla de crear un programa que utilice una DLL es utilizar la técnica de enlace en tiempo de carga, valiéndose para ello de una biblioteca de im-

portación creada mediante la utilidad IMPLIB. Dicha utilidad genera, a partir de una DLL o de su fichero DEF, una pequeña biblioteca de enlace estático -en este caso MENSAJE.LIB- que contiene toda la información que cualquier programa necesita para hacer uso de la DLL. Enlazando el programa usuario con esta biblioteca, y conociendo los prototipos de las funciones a importar -cosa que se hace en el ejemplo mediante el fichero MENSAJE.H-, el programa estará listo para utilizar la DLL. La sencillez del método se puede apreciar en el programa PROG1.C. En la mayoría de los casos no es necesario utilizar ninguna otra técnica, aunque ésta tiene las limitaciones propias del enlace en tiempo de carga, que se han discutido previamente. Además, como el proceso de resolución es automático y el desarrollador no tiene control sobre él, si la DLL no se encuentra, o algo va mal en la inicialización, el único efecto observable es que el programa se niega a cargar, lo que no proporciona demasiada información.

+potente = +complicado

Otra alternativa es la que se muestra en PROG2.C: el enlace en tiempo de ejecución. En la función *CargaMensajes()* se observa cómo se realiza la resolución de las referencias a la DLL por medio de llamadas al API del OS/2. En primer lugar se llama a la función *DosLoadModule()*, a la cual se proporciona el nombre de la DLL deseada. Si el sistema operativo encuentra la DLL, devuelve un descriptor del tipo HMODULE que se utilizará en lo sucesivo para referirse a dicha DLL. Posteriormente, y para realizar la resolución de la referencia, se llama a la función *DosQueryProcAddr()* utilizando como parámetros el descriptor citado y el nombre de la función que se quiere resolver. Si todo va bien, se devuelve un puntero al punto de entrada de la función deseada. A partir de ese momento, se puede utilizar la función normalmente. También existen llamadas para descargar temporal o permanentemente la DLL de la memoria, aunque no se han utilizado en este caso. Como se puede apreciar, la ventaja del enlace en tiempo de ejecución reside en que tanto el nombre de la DLL como el de las funciones son dinámicos y se pueden decidir "a programa arrancado", así como el momento en que se carga la DLL en memoria. Además, si algo va mal en el proceso de enlace, podremos saber el porqué con mayor exactitud. El inconveniente es que resulta mucho más complicado que el método utilizado en PROG1.C.

El enlace dinámico es más complicado que el enlace estático

POR FIN, LA EJECUCION

Para ilustrar todo lo visto hasta ahora, examine el efecto de los programas PROG1.C y PROG2.C en ejecución. No se olvide primero de copiar MENSAJE.DLL a alguno de los directorios incluidos en el LIBPATH; es aconsejable que alguno de dichos directorios, especificados en CONFIG.SYS, sea el directorio actual (representado, como en PATH, simplemente con un punto). Esto flexibilizará las pruebas con DLLs.

Los programas PROG1.C y PROG2.C sólo se diferencian en la técnica de enlace que utilizan. Por lo demás, hacen exactamente lo mismo: mostrar dos mensajes, esperar a que el usuario pulse una tecla y terminar. Observe lo que ocurre con el código de inicialización y terminación cuando se arrancan dos o más copias del mismo programa, por ejemplo haciendo

```
START PROG1
START PROG1
```

EL ULTIMO QUE APAGUE LA LUZ

Observe como el primero de los programas arrancados es el que muestra la cadena *'Inicializando sistema de mensajes...'*. Esto evidencia que se ha llamado a la función de inicialización global, que se encarga de cargar el fichero en memoria. Los otros dos programas no muestran esa cadena: el fichero ya estaba en memoria y no era necesario volver a cargarlo. Ahora, vaya pulsando una tecla en cada una de las sesiones abiertas para cada programa. El último programa en cerrar, y sólo el último, será el que muestre el mensaje *'Cerrando sistema de mensajes...'*. Aunque en la figura adjunta, dicho programa era además el primero en cargarse, no tiene por qué serlo.

Puede observar cómo un cambio del fichero de mensajes hace que cambien las salidas de los programas. También puede cambiar algún mnemónico para provocar un mensaje del tipo *"No encontrado"*.

YA PASO LO PEOR

Como se ha visto, la programación de DLLs pone en juego muchos conceptos avanzados de lenguaje C y de manejo del Sistema Operativo. Llegados a este punto, se ha mostrado de manera muy concentrada la mayoría de dichos conceptos. Aunque MENSAJES.DLL es una DLL muy sencilla por lo que hace, esta sencillez funcional es engañosa: su complejidad estructural hace que Vd. pueda utilizarla como esqueleto para construir DLLs de su propia cosecha y grandes prestaciones. Lo difícil ya está hecho; sólo es cuestión de rellenar los huecos. ■

MENSAJE.C

```
/************************************************
  MENSAJE.C: Biblioteca de enlace
  dinamico para la explotación de un
  fichero de mensajes
************************************************/

#define INCL_DOS
#define INCL_DOSERRORS
#define INCL_DOSMODULEMGR
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "mensaje.h"


/************************************************
  Definiciones
************************************************/
#define PUBLICA _export
#define FICHERO_MENSAJES "msg.txt"
#define MAX_LONG_CLAVE 80
#define MAX_LONG_MSG   100
#define MAX_LONG_LINEA
MAX_LONG_CLAVE+MAX_LONG_MSG+
1
#define MAX_MENSAJES 256
#define DELIMITADOR '='


/************************************************
  Prototipos de funciones de definición
  externa
************************************************/
int _CRT_init(void);
int _CRT_term(void);
ULONG _System _DLL_InitTerm (HMO-
DULE hModule, ULONG ulTerminating);


/************************************************
  Prototipos de funciones privadas
************************************************/
ULONG TerminacionGlobal (void);
ULONG InicializacionGlobal (void);
ULONG TerminacionInstancia (void);
ULONG InicializacionInstancia (void);


/************************************************
  Tipos de dato
************************************************/
typedef struct _mensaje
{
   UCHAR clave[MAX_LONG_CLAVE+1];
   UCHAR msg[MAX_LONG_MSG+1];
} MENSAJE;


/************************************************
  Variables globales (compartidas)
************************************************/
#pragma data_seg(COMUN)

ULONG num_mensajes;
```

```
MENSAJE mensajes[MAX_MENSAJES];
USHORT usuario=0;

#pragma data_seg()


/************************************************
  Funciones p£blicas
************************************************/
UCHAR *Mensaje(UCHAR *Clave)
{
   MENSAJE *m;
   ULONG i;
  for (i=0,m=mensajes;
        i<num_mensajes;++m,++i)
  {
     if (!strcmp(m->clave,Clave))
        break;
  }
  if (i==num_mensajes)
     —m;  // Si no se encontr¢ el mensaje,
           // utilizar el último mensaje del
           //  fichero
   return m->msg;
}


/************************************************
  Funciones privadas
************************************************/

ULONG _System _DLL_InitTerm
(HMODULE hModule, ULONG ulTerminating)
{
   if (ulTerminating)          // Terminación
   {
      —usuario;
      if (!TerminacionInstancia()) // De instancia
         return FALSE;
      if (usuario==0)
      {
      if (!TerminacionGlobal())    // Global
         return FALSE;
      }
      if (_CRT_term())
         return FALSE;
   }
   else                        // Inicialización
   {
      if (_CRT_init())
         return FALSE;
      ++usuario;
      if (usuario==1)
      {
      if (!InicializacionGlobal())  // Global
         return FALSE;
      }
      if (!InicializacionInstancia()) // De instancia
         return FALSE;
   }
   return TRUE;
}


ULONG TerminacionGlobal (void)
{
```

```
#ifdef TRACK_INITTERM
   printf ("Cerrando sistema de
            mensajes...\n");
#endif
   free(mensajes);
   return TRUE;
}


ULONG InicializacionGlobal (void)
{
   FILE *fi;
   MENSAJE *pmsg;
   ULONG rc;
   UCHAR l[MAX_LONG_LINEA+1];
   UCHAR *p;
   USHORT long_clave;

#ifdef TRACK_INITTERM
   printf ("Inicializando sistema de
            mensajes...\n");
#endif
   rc=FALSE;
   fi=fopen(FICHERO_MENSAJES,"r");

   if (fi==NULL)
      return rc;  // No se pudo abrir el
fichero

   for (num_mensajes=0,pmsg=mensajes;;
        ++num_mensajes,++pmsg)
   {
      if (!fgets(l,MAX_LONG_LINEA,fi))
      {
         rc=TRUE;
         break;  // Error de lectura
                 // de fichero
      }
      l[strlen(l)-1]='\0';
      p=strchr(l,DELIMITADOR);
      if (!p || p==l)
         break;  // Delimitador
                 // no encontrado
      *p='\0';
      strcpy(pmsg->clave,l);
      strcpy(pmsg->msg,p+1);
   }
   return rc;
}


ULONG TerminacionInstancia (void)
{
#ifdef TRACK_INITTERM
   printf ("Terminación de instancia...\n");
#endif
   return TRUE;
}


ULONG InicializacionInstancia (void)
{
#ifdef TRACK_INITTERM
   printf ("Inicialización de instancia.
   Usuario número %d..\n",usuario);
#endif
   return TRUE;
}
```

PROG2.C

```
/*************************************************

   PROG2.C: Programa que hace uso de
   la DLL de  mensajes mediante enlace
   en tiempo de ejecución

*************************************************/

#define INCL_DOS
#define INCL_DOSERRORS
#define INCL_DOSMODULEMGR
#include <os2.h>
#include <stdio.h>


/*************************************************
  Definiciones
*************************************************/

#define LONG_MODULO_ERROR 80


/*************************************************
  Prototipos de funciones
*************************************************/

void main (void);
USHORT CargaMensajes(void);


/*************************************************
  Variables globales
*************************************************/

UCHAR * (*Mensaje) (UCHAR *Clave);


/*************************************************
  Funciones
*************************************************/


void main (void)
{
   if (CargaMensajes())
     return;
   printf ("%s\n",Mensaje("BIENVENIDA"));
   printf ("%s\n",Mensaje("PULSARTE-
CLA"));
   _getch();
}


USHORT CargaMensajes (void)
{
   HMODULE h;
   USHORT rc;
   UCHAR ModuloError[81];
   rc=DosLoadModule(ModuloError,80,
   "MENSAJE",&h);
   if (rc)
   {
      printf ("No se pudo cargar la DLL
        'Mensaje'. Información adicional:
        %s'\n",ModuloError);
      return 1;
   }
   rc=DosQueryProcAddr(h,0L,
      "Mensaje",(PFN *)&Mensaje);
   if (rc)
   {
      printf ("No se encontró la función
        'Mensaje' en la DLL.\n");
      return 1;
   }
   return 0;
}
```

PROG1.C

```
/*************************************************

   PROG1.C: Programa que hace uso de
   la DLL de  mensajes mediante una
   biblioteca de importación

*************************************************/


#define INCL_DOS
#define INCL_DOSERRORS
#define INCL_DOSMODULEMGR
#include <os2.h>
#include <stdio.h>

#include "mensaje.h"


/*************************************************
  Prototipos de funciones
*************************************************/

void main (void);

/*************************************************
  Funciones
*************************************************/


void main (void)
{

 printf ("%s\n",Mensaje("BIENVENIDA"));
   printf ("%s\n",Mensaje("PULSARTECLA"));
   _getch();

}
```

MAKEFILE

```
# Fichero MAKEFILE para la demostración
# del uso de bibliotecas de enlace dinámico


CFLAGS=/Ms /Fm+ /W3 /Ss /Ti+ /B"/NOE

/BASE:0x10000"


DLLOPT=/Ge- /DTRACK_INITTERM


CC=icc


all:mensaje.dll prog1.exe prog2.exe


mensaje.dll:mensaje.c mensaje.h mensaje.def
 $(CC) $(CFLAGS) $(DLLOPT) mensaje.c men-
saje.def
 implib mensaje.lib mensaje.dll


prog1.exe:prog1.c mensaje.h
 $(CC) $(CFLAGS) prog1.c mensaje.lib


prog2.exe:prog2.c mensaje.h
 $(CC) $(CFLAGS) prog2.c

_
```

MENSAJE.DEF

```
LIBRARY MENSAJE INITINSTANCE

DESCRIPTION 'DLL de explotaci¢n de
fichero de mensajes'

PROTMODE
DATA MULTIPLE NONSHARED

SEGMENTS

   COMUN CLASS 'DATA' SHARED

EXPORTS
   Mensaje

_
```

MENSAJE.H

```
/*************************************************

  MENSAJE.H: Biblioteca de enlace diná-
mico para  la explotaci¢n de un fichero de
mensajes

*************************************************/

/*************************************************
  Prototipos de funciones públicas
*************************************************/


UCHAR *Mensaje(UCHAR *Clave);
```

PROG1_1.OUT

```
Inicializando sistema de mensajes...
Inicializaci¢n de instancia. Usuario n£mero
1..
Bienvenidos a la demostraci¢n de enlace
din mico

Pulse una tecla para terminar...
Terminaci¢n de instancia...

Cerrando sistema de mensajes...

_
```

PROG1_2.OUT

```
Inicializaci¢n de instancia. Usuario n£mero
2..
Bienvenidos a la demostraci¢n de enlace
din mico

Pulse una tecla para terminar...
Terminaci¢n de instancia...

_
```



BOLSA DE TRABAJO

BOLSA DE TRABAJO

Comienza la sección dedicada a la demanda de empleo. Como novedad y para que sea realmente útil esta página, todas las empresas que necesiten expertos programadores, pueden publicar de forma gratuita sus ofertas de empleo. Para establecer el contacto con algún anunciante, diríjase a SOLO PROGRAMADORES, mencionando la referencia de la oferta o demanda, bien por teléfono, fax o correo.

- **Fco. Javier Solís**. Toledo. REF: 1
  - Técnico Especialista en Informática de Gestión.
  - Unix, redes locales, SQL, Informix, Oracle, C y Clipper.
  - Inglés.

- **Alejandro Marqués.** León. REF: 2
  - Diplomado en Informática. Esp. Sistemas.
  - C, C++, Unix, PC.
  - Inglés.

- **Joaquín Herreros.** Madrid. REF: 3
  - FP2 Informática de Gestión.
  - COBOL, C, Clipper, Metodologías Warnier y Bertini.
  - Inglés, Francés.

- **Raúl Ortega.** Madrid. REF: 4
  - Técnico de Sistemas por la E.S.I.
  - Redes locales, Cobol, C, RPG II, Guru, C.L., AS-400, Informix, SQL.
  - Inglés.

- **Sorni Ferrer.** Valencia. REF: 5
  - Diplomado en Informática de Gestión.
  - Unix, SQL, Pascal, BASIC, bases de datos.
  - Inglés, Francés.

- **Antonio Gómiz.** Murcia. REF: 6
  - Licenciado en Matemáticas. Esp. Estadística, C.N. e I.Op.
  - C++, Windows, redes locales, programación en tiempo real.
  - Inglés.

- **Antonio Pérez.** Madrid. REF: 7
  - Diplomado en Ciencias Geológicas.
  - Visual C, Unix, Xwindows, ODBC, CASE.
  - Inglés.

- **Francisco Cantos.** Córdoba. REF: 8
  - Técnico Especialista en Programación de Gestión.
  - Adabas, Natural, Unix, redes locales, Clipper, COBOL, Niros, AOS-VS.
  - Inglés.

- **Emilio Postigo.** Madrid. REF: 9
  - Técnico Especialista en Informática de Gestión.
  - Btrieve, COBOL, Dbase III, C++.
  - Inglés y Francés.

- **Fernando Martínez.** Madrid. REF: 10
  - Ingeniero Superior Naval.
  - Fortran, Clipper, BASIC, Autocad 12.
  - Inglés.

- **José Jiménez.** Madrid. REF: 11
  - Ingeniero Superior de Telecomunicación.
  - Ada, Pascal, Prolog, Clipper, Visual BASIC, Unix, OS/2.
  - Inglés, Francés, Alemán.

- **Laura Bachiller.** C. Real. REF: 12
  - Ingeniero Técnico Informático. Esp. Sistemas.
  - Redes locales, Marketing,
  - Inglés, Francés.

- **Jordi Sais.** Girona. REF: 13
  - Licenciado en Informática.
  - C++, C, Pascal, redes locales, VAX/VMS, Unix, SQL, Lisp.
  - Inglés.

- **Alejandro Fdez**. Sevilla. REF: 14
  - Diplomado en Informática. Esp. Sistemas Físicos.
  - Unix, Natural, Adabas, C, Ensamblador, COBOL, Dbase VI.
  - Inglés.

- **Luis F. Arroyo.** C. Real. REF: 15
  - Ingeniero Técnico en Informática de Gestión.
  - Clipper, C, C++, Pascal, COBOL, dBase, Unix.
  - Inglés.

- **David López.** Madrid. REF: 16
  - Diplomado en Informática de Gestión.
  - Fortran, dBase III, SQL, C, Clipper, AOS/VS II, Red Novell, Unix, MVS.
  - Inglés.

- **Juan García.** L'Hospitalet. REF: 17
  - Ingeniería Técnica de Informática de Gestión (en curso).
  - Pascal, Modula-2, C.
  - Inglés.

- **Luis Escobar.** C. Real. REF: 18
  - Ingeniero Técnico Informático de Sistemas.
  - AS-400, AIX 3, redes locales, Pascal, C++, Clipper, Lisp, Prolog.
  - Inglés.

- **Alberto Martínez.** Alicante. REF: 19
  - Ingeniero Técnico en Informática de Gestión.
  - C, C++, dBase, Clipper, redes área local y extendida, X-Window, Unix.
  - Inglés.

- **Juan C. García.** Alicante. REF: 20
  - 1er Curso de Ingeniería de Telecomunicaciones.
  - BASIC, Pascal, C, dBase III, Clipper, redes locales.
  - Inglés.













MINISTERIO DE DEFENSA
DIRECCION GENERAL DE LA GUARDIA CIVIL
MINISTERIO DE CULTURA
MINISTERIO DE ECONOMIA Y HACIENDA
MINISTERIO DE INDUSTRIA, COMERCIO Y TURISMO
GENERALITAT DE CATALUNIA. DEP. SANITAT
SERVEI CATALA DE LA SALUT
BOLETIN OFICIAL DEL ESTADO

BANCO DE SANTANDER
INSTITUTO NCNAL. DE LA SEG. SOCIAL
MINISTERIO PARA LAS ADMINISTRACIONES PUBLICAS
FONDO DE GARANTIA DE DEPOSITOS
BANCO SANTANDER PUERTO RICO
BANCO ATLANTICO
PRICE WATERHOUSE
TOSHIBA

TELEFONICA
SANTANDER NATIONAL BANK
CORPORACION FINANCIERA HISPAMER
ANALISTAS FINANCIEROS INTERNACIONALES
SEGUROS OCASO
EUROSEGUROS BBV
SEGUROS ATHENA
FYCSA (ALCATEL)

CONSTRUCCIONES AERONAUTICAS (CASA)
PATENTES TALGO
SINTEL
AVIACO
AEROPUERTO DE CANARIAS
TRANSMEDITERRANEA
PUERTOS DEL ESTADO
GEC ALSTHOM
L OREAL

FASA RENAULT
RED DE CONCESIONARIOS RENAULT
REPSOL EXPLORACION
REFINERIA DE GIBRALTAR
PETROGAL
ONDA CERO RADIO
TELEMADRID









TELECINCO (GESTEVISION-TELECINCO)
UNIVERSIDAD AUTONOMA DE MADRID
UNIVERSIDAD CARLOS III DE MADRID
INSTITUTO DE EMPRESA
TELEFONICA SISTEMAS*
S.P. EDITORES*
OLIVETTI*

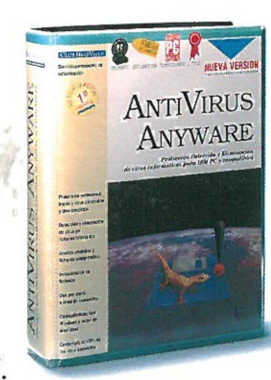
IBM*
SIEMENS NIXDORF*
DIGITAL*
BULL ESPAÑA*
FUJITSU*
INVESTRONICA*
SOFTWARE DE DIAGNOSTICO*

COMPUTER ASSOCIATES
G.P. INFORMATICA*
INFORMATICA EL CORTE INGLES*
ACTION*
GTI*
ALCATEL SISTEMAS DE INFORMACION*
INDAS

EL CORTE INGLES*
SOFTWARE DE ESPAÑA*
INFORMIX
ABBOTT CIENTIFICA
ALBILUX
ELIDA GIBBS
OXFORD UNIVERSITY PRESS

9 de cada 10 Ordenadores prefieren AntiVirus Anyware.

El resultado está a la vista. O, ¿cree que tantas Empresas pueden estar equivocadas?. Tienen razones para no estarlo.



1. MAYOR PROTECCION.
Incorpora un Sistema de Protección con una ocupación mínima en RAM. Detecta el Virus antes de que pueda introducirse e impide el uso del fichero contaminado, avisándole a través de una ventana de alarma.

2. DETECCION INSTANTANEA.
De forma rápida y precisa, analiza cualquier unidad, directorio o fichero. Comprueba si su disco duro o disquetes contienen algún Virus. Chequea ficheros (incluso comprimidos), sector de arranque, tabla de particiones y unidades de red.

3. ELIMINACION DEFINITIVA.
Elimina los Virus allí donde se encuentren, sin dañar los ficheros.

4. SERVICIO DE ACTUALIZACION PERMANENTE.
Usted puede recibir las nuevas versiones cómodamente en su domicilio o capturarlas vía modem.

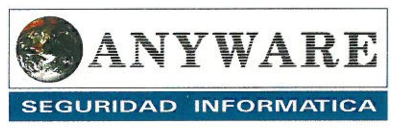
ANYWARE pone a su disposición el CLUB DE USUARIOS HELP VIRUS con una HOT LINE exclusiva para consultas, noticias, BBS, etc.








PREMIO PC MAGAZINE MEJOR UTILIDAD 1991 · RECOMENDACION PC WORLD MEJOR PRODUCTO ANTIVIRUS · RECOMENDACION PC MAGAZINE MEJOR PRODUCTO ANTIVIRUS · CATACOM 93 MEJOR ANTIVIRUS · PREMIO PC MAGAZINE MEJOR UTILIDAD 1993



Orense, 36 3º. 28020 MADRID. España.
Tel.: (91) 556 92 15. Fax.: (91) 556 14 04.

EMPRESAS E INSTITUCIONES CON LICENCIA CORPORATIVA DE ANTIVIRUS ANYWARE
(*)EMPRESAS DISTRIBUIDORAS DE ANTIVIRUS ANYWARE.